# DOMINO and NUCLEUS Software Guide

ND-820026.1 EN

Norsk Data

# *DOMINO and NUCLEUS*
# *Software Guide*

*ND-820026.1 EN*

*Send all documentation requests to:*

*Norsk Data A.S*
*Graphic Centre*
*P.O. Box 25 – Bogerud*
*N-0621 Oslo 6*
*NORWAY*

**The manual**

This manual describes DOMINO and NUCLEUS. Most of the modules are documented in separate chapters.

**The reader**

This manual is intended for maintenance personell and system developers.

**The Products**

The OS-kit consists of several software modules to be used for running and developing system software on the DOMINO IO-controllers. DOMINO is only available on ND-5000 computers with MF-Bus memory (former MPM-5), whereas NUCLEUS is also available on ND-500-II with OCTOBUS. The DOMINO controllers are based on the Motorola MC-68020 microprocessors.

The OS-kit consists of:

- DOMINOS
- DOMINO Monitor and a "gateway" to OCTOBUS (BOPCOM Server).
- DOMINO Debugger (slightly modified Symbolic Debugger)
- DOMINO OPCOM (firmware)
- NUCLEUS
- NUCLEUS Monitor

**Prerequisite knowledge**

The user should be familiar with general program development on ND computers. It is not necessary to know much about the Motorola assembly language as most of the programs can be written in PLANC.

The following objects are important for program development under DOMINO/NUCLEUS:

- SINTRAN RT-programs and ND-5000 applications
- PLANC programming language
- ND-500 Linkage Loader

**Related manuals**

| | |
|---|---|
| MPM-5 Technical Description | ND-810004 |
| DOMINO Standard Hardware | ND-814001 |
| SINTRAN III Commands Reference Manual | ND-860128 |
| SINTRAN III Real Time Guide | ND-860133 |
| SINTRAN III Monitor Calls Guide | ND-860228 |
| PLANC Reference Manual | ND-860117 |
| Symbolic Debugger User Guide | ND-860158 |
| LED User Guide | ND-860266 |
| ND-500 Loader/Monitor | ND-860136 |
| Linkage Loader User Guide & Reference | ND-860182 |

Manuals for the MC68xxx microprocessors (published by Motorola Inc)

Table of contents

## Table of appendices

## List of figures

# List of tables

# Chapter 1 Introduction

**DOMINO**

The basic idea of DOMINO is to have a range of powerful IO-controllers able to support the IO-needs for the ND-5000 CPUs. DOMINO introduces new hardware and software architecture for this purpose. DOMINO contains a standard environment for DOMINO IO-controllers, which make development easier for new applications.

**Hardware**

The manual deals with DOMINO, as seen from a software point of view. Only a short overview is given of the hardware architecture. (See the manual "DOMINO Standard Hardware" (ND-814001)).

The DOMINO controllers are connected to the common MFbus (Multi Function) memory. Each controller is able to transfer data to and from this memory (Direct Memory Access), which is the main data path. The MFbus, and all CPUs attached, support semaphore cycles to allow for process synchronizing. The MFbus has 32-bits data and address buses.

**OCTOBUS**

The OCTOBUS is a serial bus intended for sending short messages. It is mainly used for process synchronization. During initialization it passes configuration parameters. The DOMINO Monitor uses it as a communication path through the BOPCOM server.

**MFbus**

The MFbus Controller initializes the DOMINO controllers at power-up. OCTOBUS parameters and address space for the DOMINO controller in the MFbus memory are set. The very first time, this must be done by ND System Integration staff or ND service/support staff running the MFbus Controller Maintenance program.

| | |
|---|---|
| **DOMINO controller** | The DOMINO controller supports dedicated IO-processes (applications) to run within a common environment. Device-dependent hardware and software are added for each DOMINO-based development project. |
| **Local memory** | A Controller can have from 1/2 to 8 MB of local memory. There is a parity bit for each byte in the 32-bit word. |
| **Memory protection** | Memory protection is needed in the software architecture where many tasks run concurrently. The protection system in hardware supports such needs. A bus error is generated if attempting to refer an address with wrong privileges. |
| **Timers** | The MFP (Multi Function Peripheral) has four timers. One of them is used for generating clock interrupts. The MFP has also the USART for the terminal interface. |
| **Debugging tools** | Some parts of the controller are present to ease developing and maintenance. |
| **Breakpoint** | A breakpoint can be defined for each memory location. This ensures fast program execution even when running with a debugger, as checking for breakpoints is handled by hardware. Local CPU processing power is not used for breakpoints. |
| **Trace connectors** | The bus signals are available on trace connectors. A logic analyzer can be attached to the target via these. |
| **RS232 part** | An RS232 C terminal interface allows for attaching a terminal directly to the controller. |

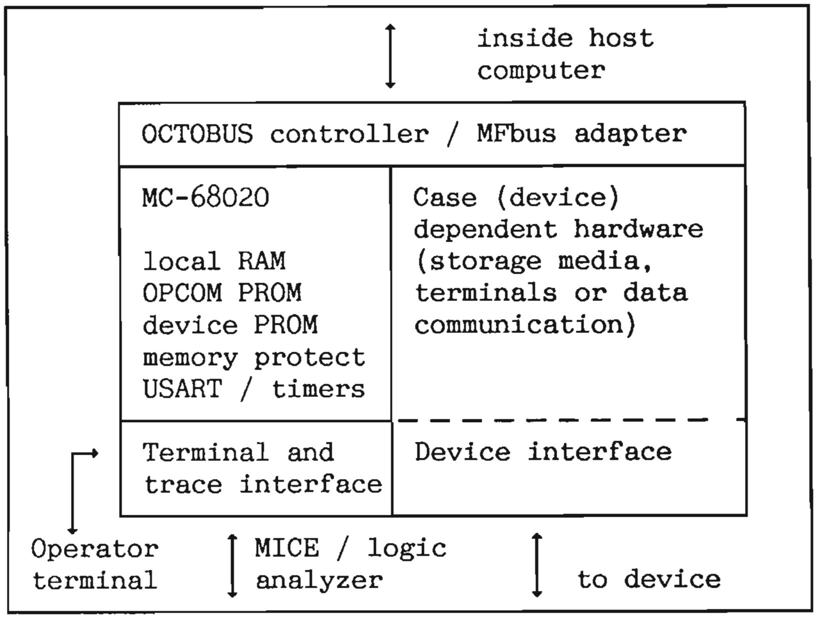



*Figure 1. DOMINO hardware components*

**DOMINOS and DOMINO monitor**

The dedicated applications handling I/O inside the DOMINO controller are run under the control of DOMINOS. DOMINOS is an operating system kernel common to all DOMINO controllers. Several applications may run concurrently as separate processes.

The DOMINO Monitor is an ordinary SINTRAN user-program, which is used for down-loading and debugging of applications in the DOMINO controllers.

Both stand-alone applications and applications controlled by DOMINOS can be run. DOMINOS and its application processes are loaded at the same time into one domain. Code (for new processes) cannot be added to the DOMINO controller while it is running.

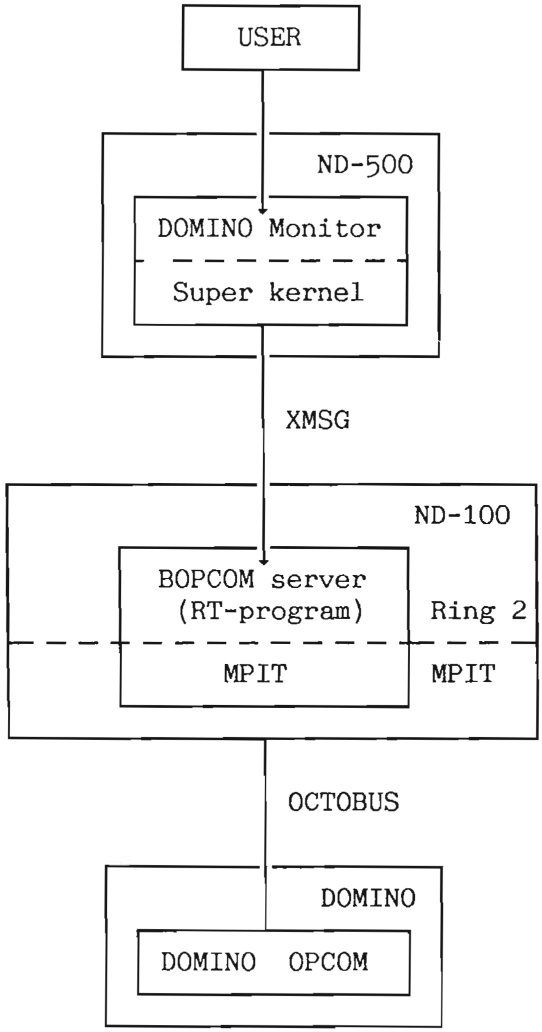



*Figure 2. DOMINO SW components*

**DOMINO communication**

The DOMINO Monitor communicates with the PROM-based OPCOM module in the controller (OPCOM means DOMINO OPerator COMmunication). It contains interrupt drivers for OCTOBUS and its local terminal interface. There is also code for performing hardware-related tests and code for execution of the commands via the DOMINO Monitor.

Several commands for debugging and maintenance are available in the DOMINO Monitor. OPCOM is mainly invisible for the programmer.

Debugging may continue even after an application has crashed, as the firmware code remains intact. The DOMINO Monitor contains hardware-related debugging commands, while the integrated DOMINO Debugger operates on source level.

The DOMINO Monitor may communicate with DOMINO OPCOM in three different ways:

- ASYL - ASYnchronous Line (terminal interface). Communication from terminal line on ND-100 to terminal line on DOMINO.
- SERVER - BOPCOM server. This is by far the most used way of communication. Both ND-100 and the DOMINO controller need an OCTOBUS station.
- MICE - Micro-In-Circuit Emulator replacing the MC68xxx processor. This is mainly used for debugging during hardware development of the controller.

**Mailbox**

A mailbox may be used in addition to terminal line and the server. The mailbox consists of a fixed part of physical MFbus memory(MPM). It must be accessible from both the DOMINO Monitor and the DOMINO controller. The data transfer becomes faster when using mailbox instead of serial transmission (terminal line and OCTOBUS).

**NUCLEUS** NUCLEUS is a library for fast message passing.

**NUCLEUS use** NUCLEUS is intended to be used only for all Norsk Data System applications requiring fast and reliable message passing between processes within one computer. The processes may for instance be one server with several clients. NUCLEUS cannot be used for communication between computers.

All processes communicating via NUCLEUS have to be within the same computer. By **computer** is meant one or several main CPUs and DOMINO controllers with access to the same physical memory and OCTOBUS.

**NUCLEUS Kernel** NUCLEUS data structure reside in shared memory (MPM), operated upon by specific rules. Parts of physical memory are reserved for the data structure used by NUCLEUS.

**NUCLEUS library** The services provided by NUCLEUS are independent of the CPU and operating system where the process is running.

**NUCLEUS monitor** The NUCLEUS Monitor is a tool for inspection of tables and queues in NUCLEUS kernel.

**Communication Concepts**

Communication between processes in NUCLEUS is based on **ports** and **messages**. Their descriptions reside in physical memory shared between the CPUs. (The NUCLEUS kernel)

**Port**

A port is an address (reference), where you can contact others, and vice versa. Ports may have a name. A port contains among other things an identification of the port owner and a pointer to received messages. Messages can be linked to a port, where they are queued in the same sequence as they arrive.

**Message**

A message is a physical buffer, which is sent (linked) between ports. A message consists of a physical buffer for data and a header containing for example a buffer descriptor and link to other messages.

# Chapter 2 DOMINO Operation

## 2.1 DOMINO Overview

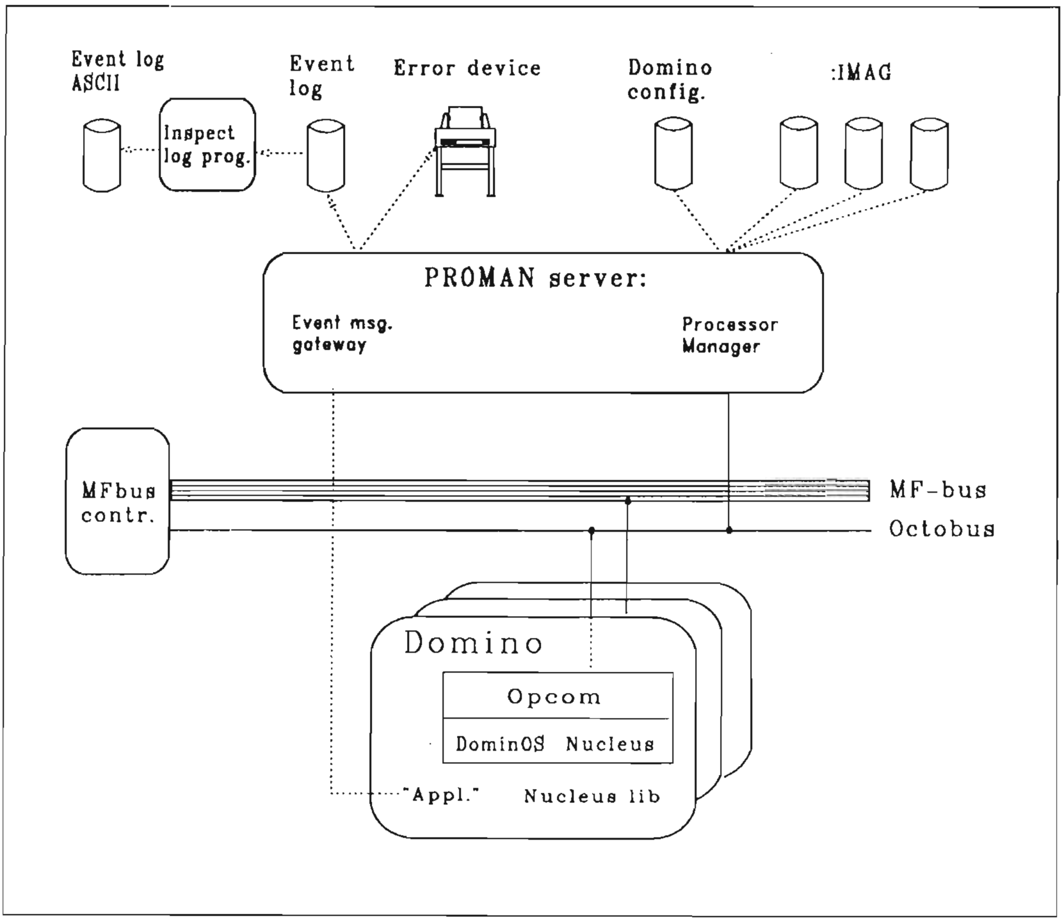



*Figure 3. DOMINO Overview*

**PROMAN**

The Processor Management server (PROMAN) is a system server running on ND-100. (See also page 21). The server is started immediately after system start and is responsible for:

- Automatic booting of DOMINO-controllers at restart/power up

When the system is running, the server provides the following services:

- Reboot DOMINO with default software on request
- Reboot DOMINO with given software on request
- Give DOMINO configuration data on request
- Terminate DOMINO-controllers on request
- Power-fail handling of DOMINO-controllers

Requests to PROMAN are sent by NUCLEUS. These requests are described in the section "Interface to configuration data and boot functions".

**PROMAN error codes**

Error codes returned from PROMAN, are found in Appendix C. See page 261-263.

## 2.2 Configuration

**Configuration** in this context, is information given to the system about the kind of hardware (DOMINO controllers) that has been installed, and about the software that can be run on it.

The DOMINO hardware consists of cards that fits into the MF-bus crate. They are recognised by the MF-bus controller. It is possible to attach a console to the MF-bus controller for configuration and maintenance purposes. The hardware part of the configuration is described in the manual "MPM-5 Technical Description" (ND-810004.01).

A minor change is made to the configuration procedure to allow for software configuration. This is the normal way of telling the system which software to run on the controllers. This method is described under "automatic configuration". The other way, done by means of a configuration file, is described under "manual configuration".

### 2.2.1 Automatic configuration

This is the normal way that the operating system is told which software to place onto the controllers.

The software for a controller is contained in an image file. See page 16.

As a general rule, a DOMINO-controller is downloaded with a predefined image according to Module Number. This is a hardwired number on each card fetched by the MF-bus controller, (module/model number). See table on page 14.

However, as there will be a need for different software to execute in several DOMINOS of the same type in the same system, changes have been made in the configuration procedure of the MF-bus controller. (MF-bus controller software version E00 or later, contained in 4 EPROMS, is prerequisite for DOMINO Operation).

This procedure is used during system integration to suit the customers needs. The configuration may also be changed by qualified service personal on-site by means of the MF-bus controller console. The software configuration is placed by the MF-bus software into the EEPROM in the back-wiring of the MF-bus.

An additional parameter may be entered during the normal hardware configuration of a DOMINO card on the MF-bus controller console.(the parameter is asked for, but is not mandatory). It is called "Basic Software Identification", and consists of a string of up to four characters, specifying the image to be downloaded.

An additional question is asked:

```
Basic Software Module identifier (4 characters):
```

All alphanumeric characters are permitted. More than 4 characters are ignored, missing characters are assumed blank. Default value is all blank (SPACE or NUL). The four byte string is saved among with the hardware parameters in the backwiring-EEPROM.

The image name is then constructed as follows:

<product id> - <hardware id> - <basic software id> : IMAG

**<product id>** Standard part to identify product relationship, the product here being the Processor Manager-server, PROMAN, which has the product prefix **PMA**

**<hardware id>** Identifier for h/w module number (4 chars). This identifier is looked up by the PROMAN program as a function of the hardware module number, see table 1 on page 14.

**<basic software id>** Software type identifier (max 4 chars). Necessary when more than one type of product runs on processors with the same module number. Examples are TCP/IP, COSMOS and SIBAS-communication, all running on Ethernet-III. This field is NOT intended to take care of version control.

If **Basic Software Identifier is omitted**, the image file name will be:

<product id>-<harware id>:imag

**Examples**

```
PMA-GRAPH:IMAG   % default image name for
                   graphical controller.

PMA-ETH3-TCPI:IMAG  % for communication TCP/IP
PMA-ETH3-COSM:IMAG  % for communication COSMOS
PMA-ETH3-SIBR:IMAG  % for SIBAS
PMA-SCSI-BDIO:IMAG  % for BDIO      etc.
```

Standard part for images

Defined at configuration (Basic Software Id)

Given by Module Number (see table)

Standard part, prefix for PROMAN = PMA

**User-area** The PMA-files are stored on the user-area of user UTILITY.

| Module Number | Hardware-id | Type of module |
|---|---|---|
| 5B | VMEI | VME-bus interface |
| 20B | IPI3 | IPI level III controller |
| 21B | SCSI | SMDE controller (SCSI) |
| 22B | ETH3 | Ethernet III |
| 23B | FPS5 | FPS-5000 controller |
| 24B | TERM | Terminal controller |
| 25B | GRAP | Graphic controller |
| 26B | MFCC | Multi function comms controller |
| 27B | VMEC | VME-bus controller |
| 30B | DMAC | MF-DMA controller |

*Table 1. DOMINO module names*

DOMINO modules, not mentioned in the table above, with Module Number in the range 5 to 76(octal), will get their hardware identifier as shown in the table below:

| Module Number | Hardware-id |
|---|---|
| 6B | 006B |
| 7B | 007B |
| . | . |
| . | . |
| . | . |
| 75B | 075B |
| 76B | 076B |

## 2.2.2 Manual configuration

It is possible to override the automatic configuration by using a configuration file.

**Configuration file**

The file must be named **PMA-CONFIG:SYMB** and placed under user SYSTEM. Configuration is not intended to be done this way under normal circumstances, but is for testing, debugging and exception cases.

An example of a configuration file is shown below:

```
12 (UTILITY)PMA-ETH3-TEST
13 (SYSTEM)TEST-DOMINO
```

**Image file name** to be downloaded. Default user is RT, and default file type is :IMAG.

**OCTOBUS Station number** to DOMINO controller (MUST be octal).

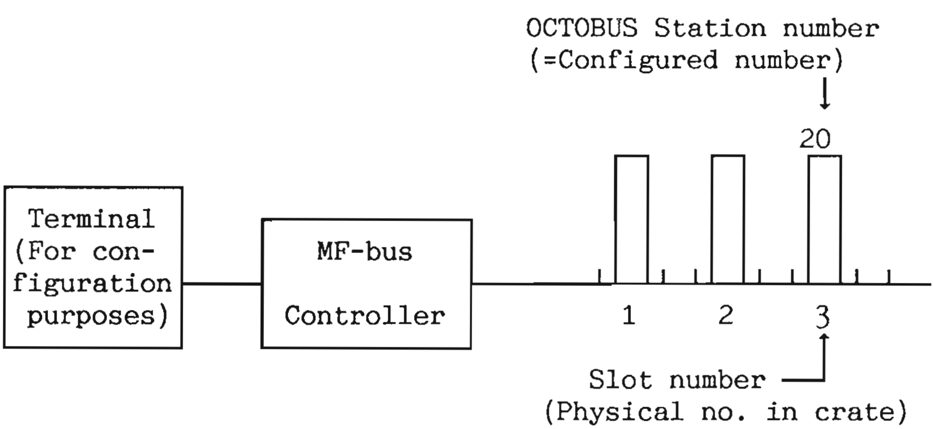



*Figure 3. DOMINO controllers in the MF-bus crate.*

## 2.3 Image files

Image files are used because they occupy less space on the disk, and are faster to place than domains.

**Image file description**

An image-file is the program and data to be placed into a DOMINO-controller, called **Basic Software Module**.

The file has a one-page header containing execution start address, bitmap and other information describing the image both in size and layout, (see figure in Appendix A,on page 250).

The rest of the file, from page one onward, is the initial content of the DOMINO's memory. Address zero in DOMINO physical memory corresponds to the start of page one on the image-file.

The image area is very often scattered, thus the file is likely to contain "holes".

The image is created by the tool "PMA-CRE-IMAGE". It takes a standard MC68xxx domain and converts it into an image file (:IMAG).

**More information**

In Appendix A, page 249-250, you will find a more thorough description of the image file.

## 2.4 Use of LEDs on DOMINO controllers

Each DOMINO controller has at least three LED (light emitting diode) in three different colors:

**Yellow**

The yellow LED is by hardware connected to the MC68K processor such that it indicates whether it is running or idle (waiting inside the STOP instruction).

**Green**

This LED indicates from release C of DOMINO OPCOM and DOMINOS, whether the application is running or not. It is lit just before the first process is started by DOMINOS and switched off when the application terminates or aborts or when DOMINO aborts. Possible user defined process management extension callable on process begin, are executed before the LED is switched on.

**Red**

The red LED is used to indicate error situations. There exists several situations when a DOMINO controller is unable to communicate via OCTOBUS. In such a situation, the controller will hang. The red LED is used to display at least some information about the reason of the fault. Different flashing patterns are used, and are interpreted as described below:



The LED is **off** all the time:
Everything seems to be OK.

The LED is **on** all the time:
This means that the selftest after reset has found some fault.The controller may used if the hardware can be avoided. (For instance the protection system).

Fast regular flashing: OPCOM receives a NAK when suspecting an ACK. ON-time = OFF-time ≈ 0.5sec. Timing for the following patterns are corresponding to this.

Regular flashing long OFF/ short ON: Something unusuable received from OCTOBUS.

Two short ON/one long OFF: Error returned when connecting to OMD.

Three short ON/one long OFF: MPROTSET returns error when initially setting up memory protection.

One log ON/two short ON: The host switch stack is empty; nobody to send to.

Three short ON/one log ON: OCTOBUS driver interface called with invalid function code.

Four short ON/one log ON: OCTOBUS driver returns error when connecting to emergency message 177B or 176B.

Seven short ON/one long ON: The path to be used is unknown in OPCOM.

Regular very slow flashing: Error returned from OCTOBUS driver when sending.

Regular flashing long ON/ short OFF: Overflow on the host switch stack.

## 2.5 DOMINO selftests

After power up, MCL and before booting, all DOMINO controllers perform a set of selftests to verify the hardware. These selftests are divided into two main groups:

- Preboot tests
- Postboot tests

The preboot tests runs in EPROM, and verifies all necessary hardware to be able to boot and run in DRAM (DOMINO Local Memory). The postboot tests consists of two parts:

- Standard postboot tests
- Device dependant postboot tests

The 'Standard postboot tests' is executed on all DOMINO controllers and test all the standard hardware parts such as MFP, interrupt system, protect system, BADAP, OBCON, etc. The 'Device dependant postboot tests' are tests specific for each type of DOMINO controllers, and tests all the special hardware functions of the different controllers.

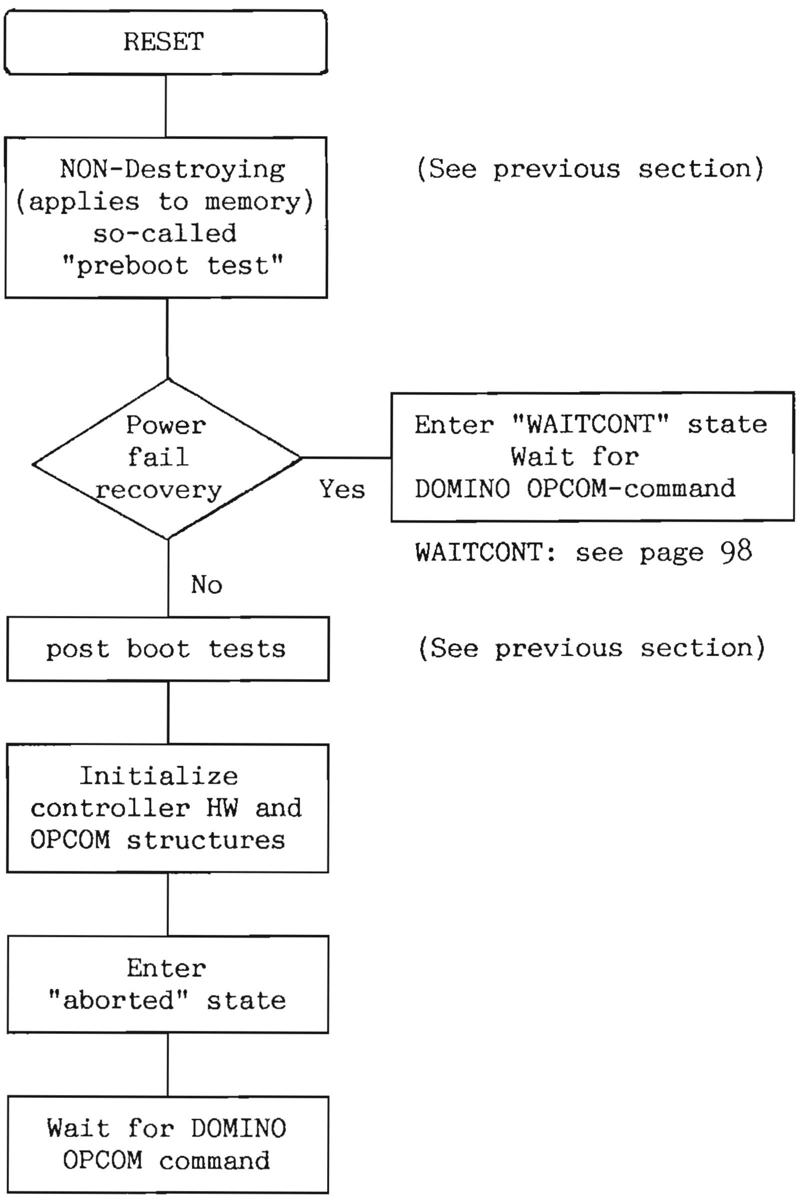
All standard tests (Preboot tests and Standard postboot tests) is located in the DOMINO OPCOM prom, while the device tests are located in the device prom.

**More information**

In Appendix B you will find a detailed description of the selftests. I.e.:

- Test numbers
- Selftest reporting to test connector
- How to use a TDF
- Exception handling in selftests
- Description and names of preboot and postboot tests

## 2.6 DOMINO reset. Algorithm



## 2.6.1 PROMAN SERVER. Algorithm

```
Check for right configuration (ND-5000)
Setup ERS-log file
Setup lamu-buffers
Initialize time-queue
Initialize internal data structures
Initialize octobus and nucleus communication
Start threads for ERS-gateway and service interface
Get configuration..
    - find all MF-bus controllers (one for each card-crate)
    - DO FOR each MF-bus-controller
    -     get crate-configuration (investigate-bank)
    -     DO FOR each DOMINO-Controller in crate
    -         get and save config data (list-configuration)
    -     ENDDO
    - ENDDO
    - get configuration from config-file (pma-config)
    - redefine configuration for those found on file
Start boot-thread for each DOMINO-Controller

DO WHILE FOREVER
    get head of time queue
    wait for event
    IF event = timeout THEN
        find tread associated with timeout event occurred
        start tread
    ELSE % Communication receive event
        IF powerfail-event THEN
            handle powerfail
        ELSE
            DO WHILE something received
                case receive type
                incase OCTOBUS
                    find thread associated with station number
                incase NUCLEUS-Service-port
                    find service thread
                Incase NUCLEUS-ERS-gateway-port
                    find ers-gateway thread
                ELSE
                    report unexpected event
                ENDCASE
                IF legal thread THEN
                    collect event information
                    start thread associated with receive event
                ENDIF
            ENDDO
        ENDIF
    ENDIF
ENDDO
```

## 2.6.2 Booting of DOMINO. Algorithm

```
Perform hard reset of DOMINO
Open image file
Get and save boot time
Perform echotest (EchoTest)
Get DOMINO-ident (IdentY), report selftest status
Perform stop (Stop)
DO
    Get block from image
WHILE blocks left in image
    Fix block in buffer
    Set mailbox pointer in DOMINO (SetBxP)
    Download block from buffer to DOMINO local memory (BxDoLd)
    Unfix block in buffer
ENDDO
CLOSE image file
Get image execution start address
put start address in DOMINO's program counter (RegMod)
Start DOMINO (Go-On)
Report DOMINO started
DO
    Start watchdog timer and wait
    Perform watchdog check
ENDDO
```

## 2.7 Event reporting and event log

ERS-reports from the DOMINO operations software are logged on a ring-file on the system disk. These reports have three different origins:

- Reports from the Processor Manager itself.
- Reports sent from OPCOM in DOMINO (sent via OCTOBUS to PROMAN). These reports are normally fatal-errors from low-level functions in DOMINO such as bus-error and unexpected traps and interrupts.
- Reports from application software running in the DOMINO (sent via NUCLEUS to PROMAN, using the "PMAreport" call.)

The ring-file buffer always contains the last ERS-messages sent by the system, and may be recalled with the "PMA-DUMP-LOG" program.

Applications in DOMINO may report standard ERS events to the Watchdog in Sintran (ERS3WD). This is done by using the routine "PMAreport" included in DOMINO Programmers Kit. (ND number: 250297).

```
ROUTINE VOID, VOID
            ( INTEGER2,       & SEC (Standard Error Code)
              BYTES POINTER)  & EventData(User parameter part
            : PMAreport       & of standard event report)
```

The routine will provide the interface to NUCLEUS, and send the report to the ERS-gateway server (part of PROMAN), which in turn will send the message to the Watchdog.

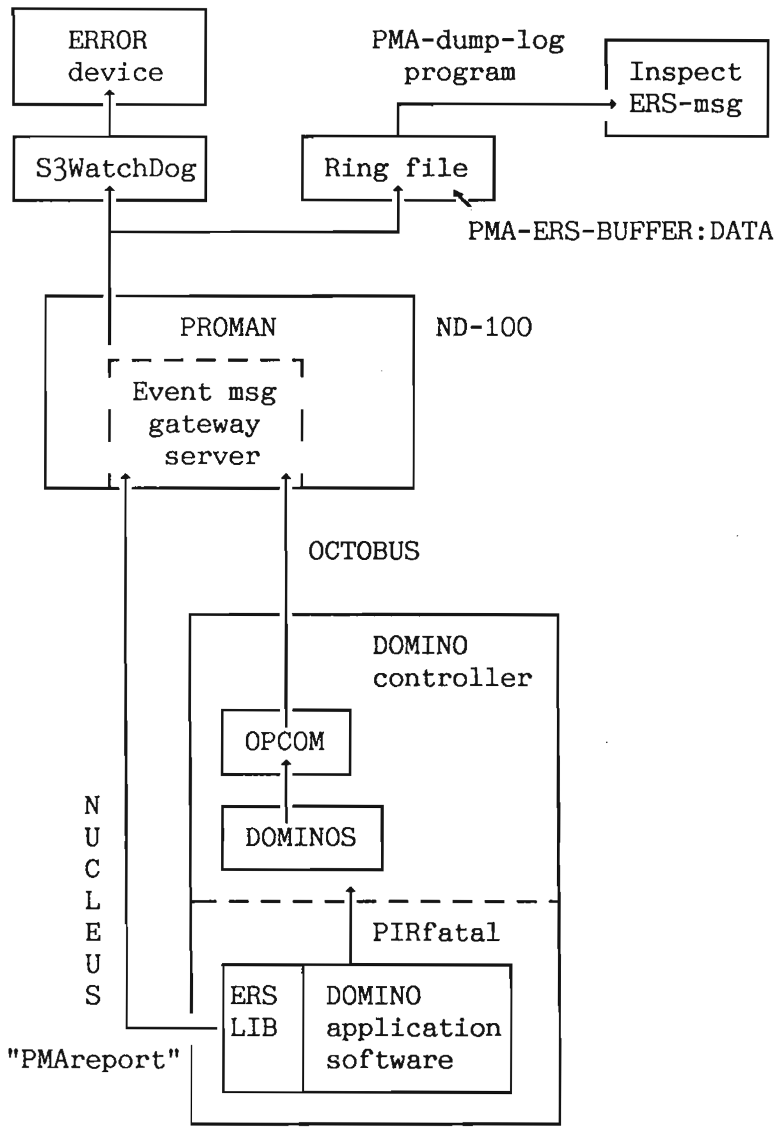



*Figure 4. Event reporting*

## 2.7.1 How to operate the event log file

The operation of the event log file (ring-file) is fully automatic. The PROMAN server will create it if it does not exist. The file may be deleted to empty the contents, or it may be recreated with new size to suit. Default size is 5 pages, which is enough to store a few hundred ERS-reports. The file must be at least two pages in size, and it must be contiguous. The file resides under user **SYSTEM** with the name **PMA-ERS-BUFFER:DATA**.

**NOTE !**

The file must not be deleted or tampered with when the PROMAN server is active. If you wish to change the size or delete it, please do this before the server is started, or immediately before restarting the system. If the server is unable to log to the ring file, a message will report this to the system error console.

### 2.7.2 How to investigate the Event Log

To investigate the Event Log, simply use the program **PMA-DUMP-LOG:PROG** supplied in the DOMINO Maintenance Kit (ND no. 211322). Start the program under user SYSTEM and give a file name on which to dump the like this:

PROCEDURE:

```
@PMA-DUMP-LOG              % call the program

Output file: "MY-LOG"      % give dump file name

Bye!
@PED MY-LOG:SYMB           % investigate the file in an editor
```

OR:

```
@PMA-DUMP-LOG "MY-LOG"

Bye!
@PED MY-LOG
```

The format dumped on the file "MY-LOG" is the same as the Watchdog server (ERS3WD) presents on the error device.

## 2.7.3 How to use the event log

The PMAreport-routine is supplied as a :NRF file. The routine must be imported into a module where it is used as follows:

```
IMPORT (ROUTINE VOID,VOID(INTEGER2,BYTES POINTER): PMAreport)
```

The file "PMA-ERS:NRF" must be included in the load session.

For details about ERS in general, see SINTRAN III Release information, L-version. (ND-860230)

## 2.8 PROMAN Service port

The server will have a NUCLEUS service port (system port) accepting requests from other system servers.

**Port name** The name of the port is **PMAservicePort.**

Several requests may be sent to the server using NUCLEUS messages. The server will acknowledge requests. Some requests may return data. The user is responsible for providing a large enough message for return of data.

> **NOTE !**
> Acknowledgements and return data will always be sent back to the request-message's home port.

## 2.9 PMA-Monitor

The PMA-Monitor provides an interactive command interface to the service port functions in the Processor Manager (PROMAN). The monitor is supplied in DOMINO Maitenance Kit (ND-211311), as a :PROG file (PMA-MONITOR:PROG).

The monitor is started by means of the command:

**@PMA-MONITOR↵**

The PMA-Monitor promts with **PMA:** whenever it is ready to accept a command.

## 2.9.1 Commands in PMA-Monitor

The following commands are direct implementations of the service port's corresponding functions. The messages "Request acknowledged" and "Request not acknowledged" are printed as a consequece of ACK or NAK from the Processor Manager.

See also the section "Interface to configuration data and boot functions". Page 34-38.

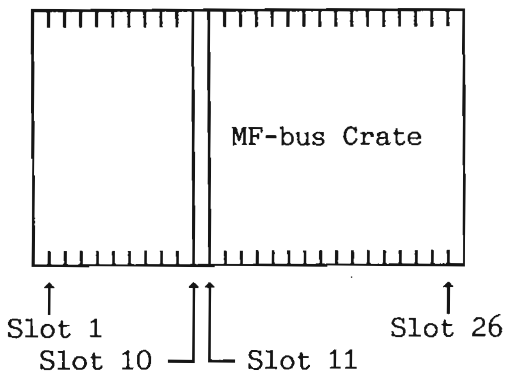
## LIST-CONFIGURATION

**Purpose** Display statistics for all the Domino controllers in a system, eg:

```
PMA: LIST-CONFIGURATION↵
SLOT 11 : Crate id 3 Octobus station 13B ---> SCSI CONTROLLER
          Module 21B Model 0B Print A Eco B
          Image file:  "(UTILITY)PMA-SCSI-BDIO"
          Boot status: Domino started
          Boot time:   1988-08-28 21:09:22
SLOT 10 : Crate id 3 Octobus station 12B ---> ETHERNET III
                                              CONTROLLER
          Module 22B Model 0B Print D Eco D
          Image file:  "(UTILITY)PMA-ETH3-TCPI"
          Boot status: Domino started
          Boot time:   1988-08-28 21:09:22
PMA:
```

The output from the LIST-CONFIGURATION command is explained on the following pages.

In this example, **Two** DOMINO controllers are present in the system.




Description of output from the LIST-CONFIGURATION command:

**SLOT** The slot location in a card crate where the DOMINO is installed.

**Crate id** DOMINO controllers reside in a card crate (card bank). **Crate id** is an unique identifier of the card crate's position. If your system has just one MF-bus crate, you may ignore this parameter. If there are more than one MF-bus bank, it is useful to know that the Crate id is actually the station number of the MF-bus controller in the Crate/Bank in question.

**OCTOBUS station** The OCTOBUS station number to the DOMINO controller.

**Module** Hardware Module Number tell what kind of card (type of DOMINO in this context) that is present in the slot. This is the origin for determining the hardware identifier in the image name, (see section about Automatic configuration, page 11).

**Model** Hardware model number.

**Print/Eco** Engineering Change Order level is an official code for the status of hardware modifications performed on the card.

**Image file** The image file name currently used (Basic Software Module). Normally this file name is the default one, or from the configuration file (PMA-CONFIG). It may also be the image name given in a RECOVER or LOAD command.

**Boot status** Tells the status of a DOMINO controller as the Processor Manager (PROMAN) sees it. These states may be one of the following:

| STATE | MEANING |
|---|---|
| Undefined state | No operation yet performed |
| Booting | Initial booting in progress |
| Rebooting | Rebooting in progress |
| Domino started | Program in Domino controller started |
| Error received from Opcom | Fatal error occurred in DOMINO |
| Booting aborted | Booting, rebooting or load aborted due to error in load |
| Terminated | Domino controller terminated due to request |
| Image placed | Image place performed |

**Boot time** The time and date when the last boot, reboot or load started.

## REBOOT-DOMINO

**Purpose** Reloads and starts the controller using the default image, or the one given in the configuration file (PMA-CONFIG:SYMB).

**Parameter** <station number>= OCTOBUS station number to DOMINO controller (default octal)

The request is acnowledged if the station number is known by the system ie. the controller is configurated.

## RECOVER-DOMINO

**Purpose** Reloads and starts the controller using the given image.

**Parameters** <station number>= OCTOBUS station number to DOMINO controller (default octal)

<image file> = Name of image file, default file-type is :IMAG, default user is RT

The request is acknowledged if the station number is known by the system ie. the controller is configurated. The syntax of the filename and the presence of the file is not checked at this point.

## TERMINATE-DOMINO

**Purpose** To stop the DOMINO controller.

**Parameters** <station number>= OCTOBUS station number to DOMINO controller (default octal)

The following functions are performed, in listed sequence:

1. Opcom stop
2. Nucleus close on behalf of controller, (releases all Nucleus resources held by controller).
3. Hard reset (selftests starts)

## LOAD-DOMINO

**Purpose** Reloads and starts the given controller using the given image. (Same as RECOVER-DOMINO command, except that the final GO command is not issued).

**Parameters** <station number>= OCTOBUS station number to DOMINO controller (default octal)

<image file> = Name of image file, default file-type is :IMAG, default user is RT

The image will be placed ready to run in the controller. Boot status in LIST-CONFIGURATION (page 31) will take value **Image placed** when the function has been performed.

## 2.10 Interface to configuration data and boot functions

See also the PMA-MONITOR on page 28. A type-definition may be found on the file:

PMA-SERVICE-COM:DEFS

The command specifications are given below:

## LIST CONFIGURATION

**Request**

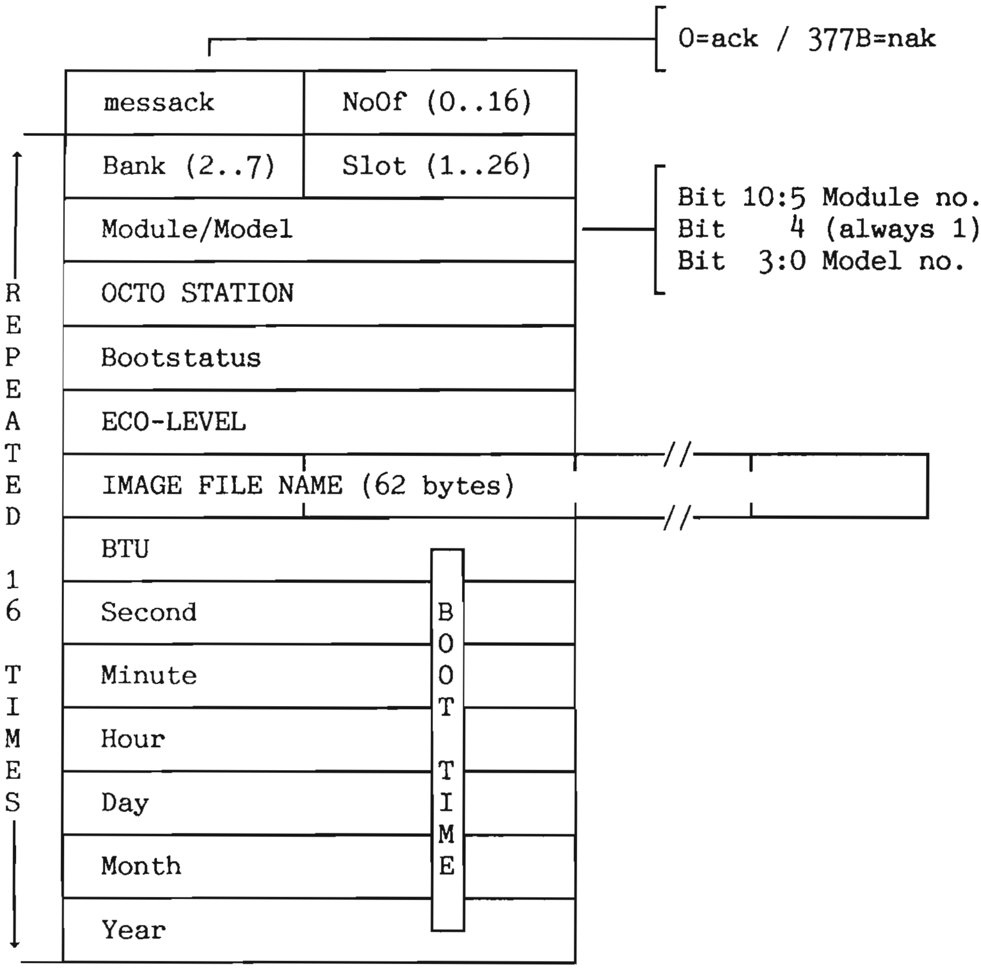
```
TYPE tConfigRequest = RECORD PACK
    BYTE: PMcommand  % = 1 for config request
ENDRECORD
```

| 1 |
|---|

Size: 1 byte

**Response**

```
TYPE tConfigEntry = RECORD PACK
 BYTE:   BankIndex, SlotIndex % range = 2..7
                              % and 1..26
  INTEGER2:  ModuleModel      % (see MF-bus controller
                              % documentation)
  INTEGER2:  OctStation       % OCTOBUS station number
  INTEGER2:  BootStatus       % (see below)
  INTEGER2:  CEecoLevel       % Module's ECO-level
                              % ( 0 if undefined)
  BYTES:     ImFileN(0:61)    % Image file name
  INTEGER2 ARRAY : BootTime(0:6) %Time of last boot
 ENDRECORD

 TYPE tConfigResponse = RECORD PACK
     BYTE: PMCRmessack, NoOfDOMINOes
     tConfigEntry ARRAY: ConfigEntry(0:15)
 ENDRECORD
```




Size: 2 + NoOfDOMINOes * 86 bytes

## BootStatus

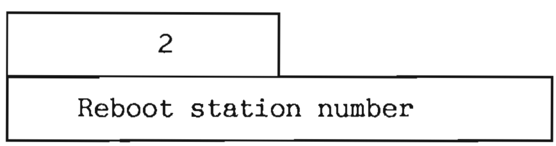
Bootstatus is a enumeration value telling the status of a controller, these are:

| | |
|---|---|
| 0 = pmUndef | Found in configuration, no action yet performed |
| 1 = pmBooting | Initial booting in progress |
| 2 = pmRebooting | Reboot in progress due to request |
| 3 = pmStarted | Controller has been started after boot/reboot |
| 4 = pmError | An error has been received from Opcom after start |
| 5 = pmAborted | Boot or reboot aborted due to error |
| 6 = pmTerminated | Controller has been terminated due to request |

## REBOOT

**Request**

```
TYPE tReBootRequest = RECORD PACK
    BYTE:      PMcommand  % = 2 for reboot request
    INTEGER2: ReBootStation
ENDRECORD
```




Size: 3 bytes

The parameter is the OCTOBUS station number of the DOMINO card you wish to reboot.

**Response**

```
TYPE tReBootResponse = RECORD PACK
    BYTE: PMRBRmessack ENDRECORD
```

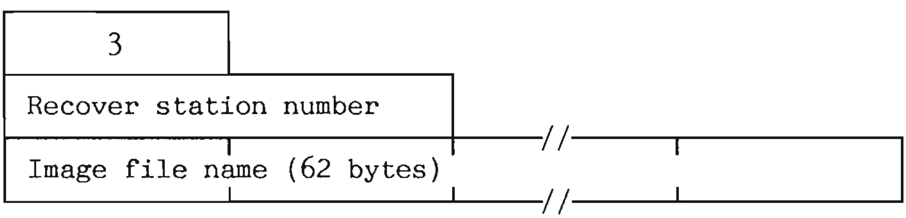

Size: 1 byte

**NOTE !** An acknowledgement here means that the controller is found in the configuration, and that a reboot is in progress. It is not an acknowledgement that a reboot has performed satisfactorily. One way confirming this is by polling the list-configuration function and watching the "BootStatus" field. A better way is to put this in the design on a higher level by sending acknowledgement from the DOMINO software when it wakes up in the controller.

## RECOVER

**Request**

```
TYPE tRecRequest = RECORD PACK
    BYTE:       PMcommand  % = 3 for recover request
    INTEGER2:   RecStation
    BYTES:      RecImageName(0:61)
ENDRECORD
```




Size: 65 bytes

**Response**

```
TYPE tRecResponse = RECORD PACK
    BYTE: PMRECmessack ENDRECORD
```


Size: 1 byte

**NOTE !** See note for Reboot.

# TERMINATE

**Request**

```
TYPE tTermRequest = RECORD PACK
  BYTE:      PMcommand  % = 4 for terminate request
  INTEGER2: TERMstation
ENDRECORD
```

| 4 | |
|---|---|
| Terminate station number | |

Size: 3 bytes

**Response**

```
TYPE tReTermResponse = RECORD PACK
    BYTE: PMTERMmessack
ENDRECORD
```

| messack | 0=ack / 377B=nak |
|---|---|

Size: 1 byte

# Chapter 3 DOMINO Monitor

This chapter describes the commands in the DOMINO Monitor. The purpose of the DOMINO Monitor is to debug and maintain DOMINO IO-controllers. The program supervises the DOMINO controller through the OPCOM module inside the target, or via an inter-circuit emulator (MICE-II).

The descriptions of the commands are grouped into sections according to function.

**Starting**

The DOMINO Monitor is a program that can be run in the ND-500/5000 computers. It can be started as follows:

```
@ND-500-MONITOR DOMINO-MONITOR↲

DOMINO-MONITOR Version C of: <Month Day>, <Year>
               Entered: <Month Day>, <Year>. Time: <Ho:Min>
DM: HELP↲
    Command:  //
    ........
DM: EXIT↲
DOMINO-MONITOR session terminated at:
                                  <Month Day>, <Year>. Time
```

**Prompt**

The DOMINO Monitor prompts with **DM:** whenever it is ready to accept a command.

**Notation**

When describing the commands available in the DOMINO Monitor, the following rules apply:

- All parameter names are enclosed in <> brackets.

- If a parameter that is asked for has a default value, the default value is enclosed within slashes //
- The names of optional parameters are enclosed in () brackets.
- If more than one value must be specified, the right bracket is followed by three dots, as in **<Parameters>**...

**Command entering**

All commands, domains and file names may be abbreviated as long as they are unambiguous. The abbreviation rules are as for SINTRAN. The full range of SINTRAN editing characters is available.

The DOMINO Monitor will prompt for missing default parameters.

**ESCape**

A Command and parameter collection can be aborted by pressing ESCape. The user returns to the command level in the Monitor. Command execution may also be interrupted in this way. Special NOTIS keys generating ESCape characters are therefore also harmless to the program.

**@**

If the first character of a command line is **@**, the rest of the line is taken to be a SINTRAN command. The command is checked before being sent to SINTRAN. This safeguards against starting another program unintentionally and thus causing automatic termination of the Monitor.

**&**

The character **&** means that the input line is continued on the next line.

**Radix**

Numeric arguments may be given in octal, decimal or hexadecimal format. The default radix is octal, but it may be changed by use of the **MAIN-FORMAT** command. A trailing **B** (octal) **D** (decimal) or **H** (hexadecimal) may override the current format except if it is hexadecimal. Hexadecimal numbers must start with a digit.

## 3.1 Miscellaneous commands

This command group is general in the sense that it is not related to any function of the DOMINO controller. Some of these commands affect the program environment.

### 3.1.1 EXIT

**Purpose** Terminates the execution of the DOMINO Monitor. Note that EXIT cannot be used within a macro. This command is used for releasing reserved resources (for example the DOMINO controller). All breakpoints - if any - are released, and if possible the control is given back to the Processor Manager.

### 3.1.2 HELP

HELP <Command>

**Purpose** All commands matching **<Command>** will be written together with their parameters to the output file.

**Command** Any command abbreviation, ambiguous or nonambiguous. Default is all commands.

**Parameters** Note that **HELP** may also be used for some parameters to obtain a list of legal choices.

## 3.1.3 SET-ABORT-BATCH-ON-ERROR

SET-ABORT-BATCH-ON-ERROR <ON/OFF: /ON/ >

**Purpose** When the DOMINO Monitor is invoked from a batch job, it usually does not make sense to continue after an error has occurred. The Monitor therefore, by default, aborts the batch job in error situations. This command is used for changing this condition. Non-critical sequences in a batch job can ignore the error conditions by using this command.

**ON/OFF** ON: the batch job should be aborted after any error.
OFF: only the current command should be ignored after error. Error messages are still output to the batch output file. This is similar to interactive execution mode.

## 3.1.4 CC

CC <any text string>

**Purpose** This command is for writing comments in a batch or mode job. It does not affect the DOMINO Monitor.

## 3.1.5 COMPUTE

COMPUTE <Expression /O/ >

**Purpose**

Evaluate and display the value of a simple arithmetic expression. The result is displayed in octal, decimal and hexadecimal format. Negative numbers are shown as two's complement for the octal and hexadecimal format.

Operations available are addition (lowest priority), subtraction, multiplication and division (highest priority). Parentheses may be used to force parts of the expression to be evaluated out of the normal priority sequence. There are no practical limitations to the number of nesting levels allowed. Unary plus and minus, real numbers and exponents are not implemented.

**Example**

```
DM: COMPUTE 1+2-3*4/(5+6-7*8D/9D)↵
          1B             1            1H
```

## 3.1.6 NEW-USER-CONTEXT

**Purpose** Changes the current SINTRAN user-area without losing any context within the DOMINO Monitor. This is particularly useful for getting necessary access to files on several user-areas. The command is for security reasons restricted to users who originally have been logged in as user SYSTEM.

**User** The name of the new SINTRAN user-area.

Example

```
DM: NEW-USER-CONTEXT DOMAINS-500↲
Now entered as user: DOMAINS-500
```

## 3.1.7 OUTPUT-FILE

OUTPUT-FILE <File name> /TERMINAL/ >

**Purpose** This command is used for directing the information stream from the DOMINO Monitor to a file. Initially this information appears on the user's terminal. Commands, parameter prompts and error messages will continue to appear on the terminal after switching. The <File name> is used as output file until **EXIT** or a new **OUTPUT-FILE** command is given.

**File name** The name of the file where output is desired. A new file can be created by giving the name within double quotes ("). Default file type is :SYMB.

## 3.2 Communication commands

These commands are related to establishing communication with the target, and for inspecting and altering parameters describing communication behaviour.

The DOMINO Monitor has to access the DOMINO controller via communication media. This can be achieved in three ways:

- Through a terminal line to an inter-circuit emulator (MICE-II). The logical name for this path is **MICE**.
- Through a terminal line to the OPCOM module running inside the controller (ASYnchronous Line). The logical name for this path is **ASYL**.
- Through BOPCOM SERVER to the OPCOM module. The logical name for this path is **SERVER**.

The terminal lines are treated as files, so they must be defined as peripheral files in SINTRAN. The path which is used for performing commands at the moment is displayed within parentheses when the DOMINO Monitor prompts for a command. This path is called the **current path**.

A command may be prefixed with a path-name inside parentheses. The path given will then be used for this command, but the current path is restored after the command is performed.

Example

```
DM(SERVER): (ASYL)LOOK-AT-STACK↲ %command performed on ASYL
DM(SERVER): LIST-MICE-PARAMETERS↲%affects only DOMINO-MONITOR
```

## 3.2.1 OPEN-PATH

OPEN-PATH <Path name /SERVER/ >,<station number>

**Purpose**

It opens the path associated with **<Path name>**, and an attempt will be made to connect to the target. This command must be given before any communication between a target and the DOMINO Monitor can start. Information about whether this has succeeded or not is displayed. The opened path is used as the current path for subsequent communication with the target.

At the most one path of each type can be opened at the same time. This is to permit several communication media to reach the same target without losing any opened path.

**Path name**

The logical name of the path. If the <Path name> is **MICE** or **ASYL** this takes place via the peripheral file. If the <Path name> is **SERVER**, it takes place via the Bopcom server to the given station number.

**Station number**

Station number associated with the given path.

**Example**

```
DM: OPEN-PATH SERVER 30↵
Connected to MC68020 based controller %connection established
DM(SERVER):                           %Bopcom server is
                                      %now current path
```

The figure below illustrates the path from DOMINO monitor to DOMINO controller using the BOPCOM server. Path name is SERVER.

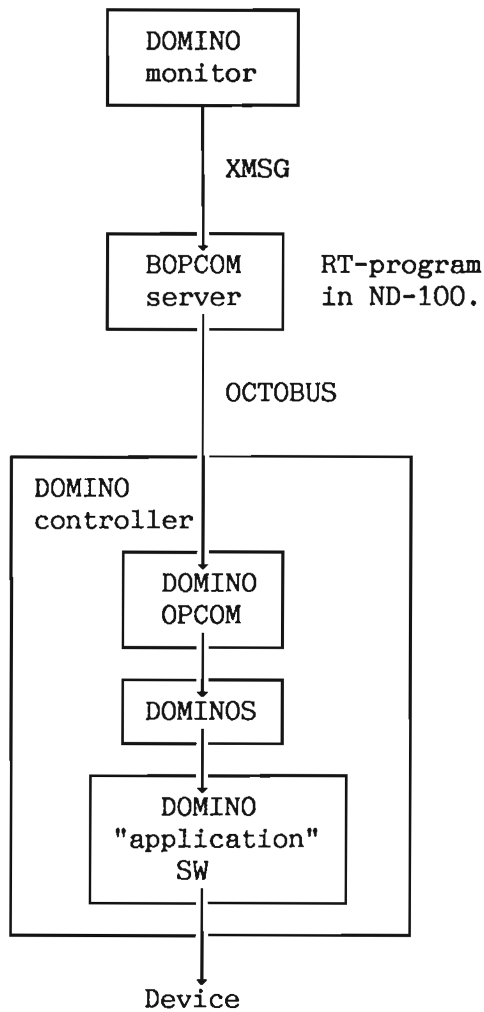



*Figure 5. SERVER path to DOMINO controller*

## 3.2.2 CHANGE-PATH

CHANGE-PATH <Path name /SERVER/ >

**Purpose** This command requires that the parameter <Path name> is open. The path will from now on be used as the current path, and <Path name> will appear between parentheses in the prompting text.

**Path name** The name of an already opened path.

Example

```
DM: OPEN-PATH SERVER 24↲
Connected to MC68020 based controller %connection established
DM(SERVER): OPEN-PATH ASYL ASYL-DISC↲ %Opening another path
Connected to MC68020 based controller %ASYL becomes current
                                      %path
DM(ASYL): CHANGE-PATH SERVER↲     % Switch back to SERVER path
DM(SERVER):
```

The figure below illustrates the path from DOMINO monitor to DOMINO controller using the ASYnchronous Line. Path name is ASYL.

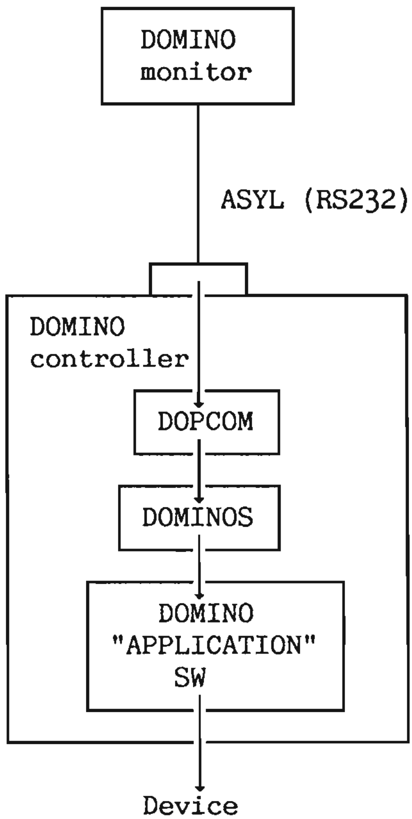



*Figure 6. ASYL path to DOMINO controller*

### 3.2.3 TEST-COMMUNICATION

TEST-COMMUNICATION <Number of times /1/ >

**Purpose**

Tests communication between the DOMINO Monitor and the target via the
standard path. The test is performed by writing and reading several
bit patterns. The communication cannot be tested via MICE.

**Number of times**

The number of times to run the communication test.

If there are no errors during the tests, two communication parameters
are reported:

- Elapsed time used on sending 100 bytes 100 times.
- Communication overhead, measured as the time used for sending 0 bytes 100 times.

If an error occurs during transmission, the following is reported:

- Bits lost, and in which direction.
- Whether data received is different from data expected or not.

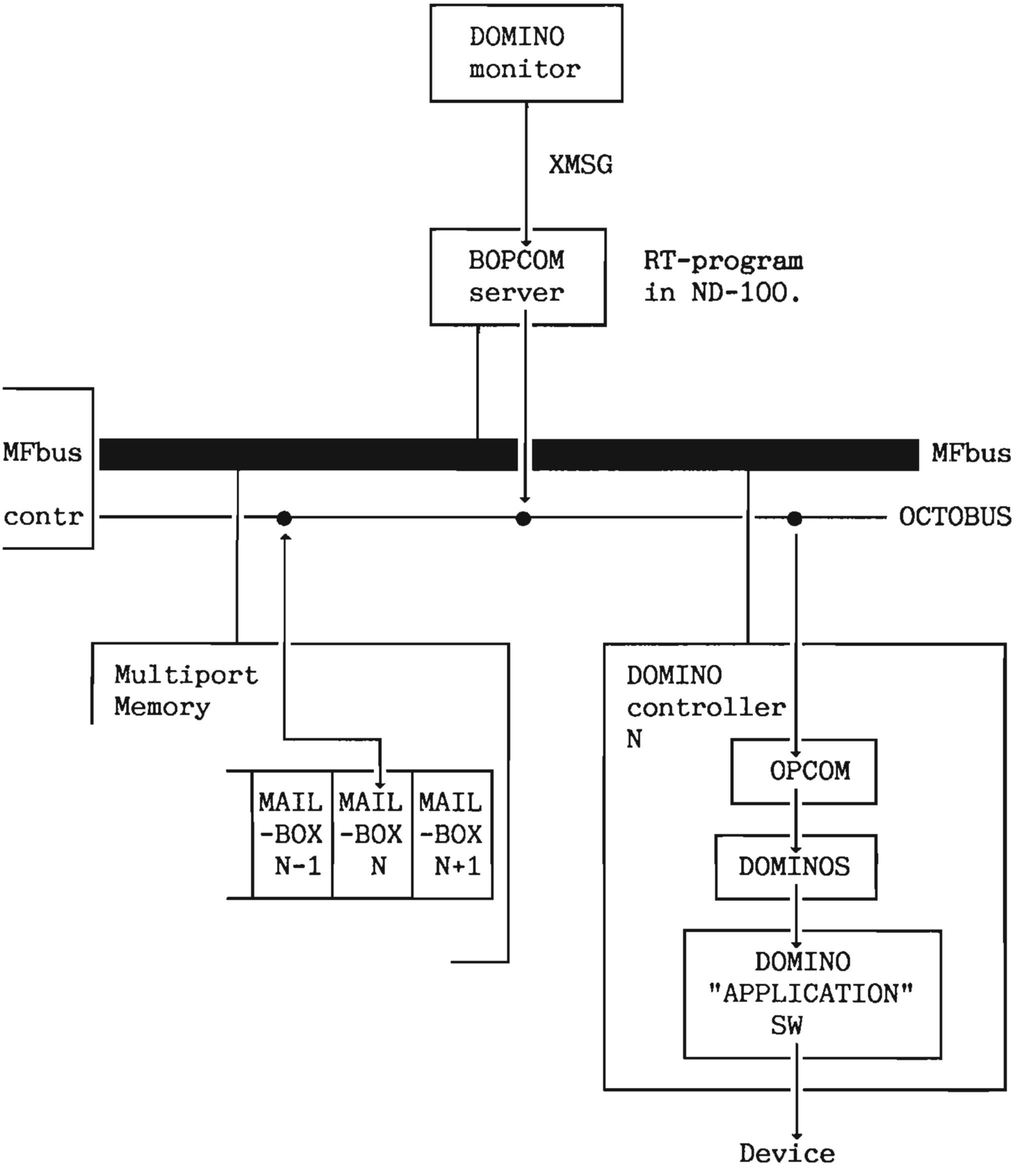



*Figure 7. SERVER path using MAILBOX*

Transparent-mode (see page 53) must be used to give input to an application running inside target (OPCOM or MICE-II).

## 3.2.4 USE-MAILBOX

USE-MAILBOX <ON/OFF: /OFF/ >

**Purpose** A mailbox can be used in addition to the terminal line or Bopcom server paths to speed-up communication. It resides in physical memory and has installation-dependent characteristics.

This command turns the use of the mailbox **OFF** or **ON** for communication in subsequent commands. Communication between DOMINO Monitor and DOMINO controller is tested if the use of it is turned **ON**. The mailbox definition remains even if the use of the mailbox is turned **OFF**.

NOTE! The **reset commands** will turn off the use of the mailbox as the DOMINO controller loses information about where the mailbox is after this command is given.

## 3.2.5 LIST-MAILBOX-PARAMETERS

**Purpose** List the parameters defining the mailbox.

Example

```
DM(SERVER): USE-MAILBOX ON↲
DM(SERVER): LIST-MAILBOX-PARAMETERS↲
Use-Mailbox                          :      ON
ND-100 page number for MF page zero :    1400B
MF page number for mailbox           :     400B
```

## 3.2.6 TRANSPARENT-MODE

TRANSPARENT-MODE (<Path name>)

**Purpose**

This command connects the user directly to the target. If the standard path is **ASYL** or **SERVER**, this is the OPCOM module. If the path is **MICE**, this is the MICE-II command processor.

All characters typed by the user go directly to the target, and the DOMINO Monitor only registers the transfer. The same applies to data sent from the target to the user. This command is terminated by typing the break character. Default value for the break character is **@** (ASCII 100B), but it may be changed with the **SET-BREAK-CHARACTER** command.

Transparent mode must be used for giving input to an application running inside the target.

**Path name**

If the optional parameter is not given, the current path is used. It has to be given if it is impossible to open a path to the target.

## 3.2.7 SET-BREAK-CHARACTER

SET-BREAK-CHARACTER <Break character /100B/ >

**Purpose**

Change the character that terminates the transparent communication mode initiated by the **TRANSPARENT-MODE** command.

**Break character**

The ASCII value of the break character. Select a value not used by the application running inside the target.

Control characters can be used as long as they are not used by the application. This is because the SINTRAN line-editing characters do not apply when the DOMINO Monitor is in transparent mode.

## 3.2.8 LIST-BREAK-CHARACTER

LIST-BREAK-CHARACTER

**Purpose**

This command displays the break character that terminates the **TRANSPARENT-MODE** command.

## 3.2.9 SET-DOPCOM-PARAMETERS

**Purpose** Several parameters concerning the communication between DOMINO Monitor and the target may be changed using this command. The unit of measure for time parameters is BTU. One Basic Time Unit is 20 ms.

**SOH-TO** SOH timeout. The maximum time to wait for receiving the Start Of Header message from the DOMINO controller after a function has been asked for.

**Succ*** Successive timeout. The maximum time to wait for reading the next data unit within a message.

**IniHold*** Initial hold. Not used in the present version of the DOMINO Monitor.

**DLoad#** Download retries. The number of unsuccessful retries to make before aborting when downloading a domain to the DOMINO controller.

**General#** General retries. The number of retries to make after a communication error (e.g. the DOMINO Monitor gives an unexpected answer, or a message has been destroyed during transmission).

## 3.2.10 LIST-DOPCOM-PARAMETERS

**Purpose** List the parameters that determine the communication behaviour between the DOMINO Monitor and the DOPCOM module. They are displayed in the same order as they appear in the **SET-DOPCOM-PARAMETERS** command.

## 3.2.11 SET-MICE-PARAMETERS

**Purpose** Several parameters concerning the communication between DOMINO Monitor and the MICE-II may be changed by using this command. The unit of measure for time parameters are either ms or BTU (1 BTU = 20 ms). Several parameters are needed for this communication, as the DOMINO Monitor requests functions on MICE by simulating operator input directly to the MICE command processor.

**Clear*** Clear timeout. The maximum time (ms) available for clearing the MICE output buffer.

**Mem*** Memory timeout. The maximum time (ms) to wait after requesting a memory location a break point change from MICE, or after giving the **GO** command.

**Reg*** Register timeout. The maximum time (BTUs) to wait after requesting a register from MICE.

**IStep*** IStep timeout. The maximum time (BTUs) to wait after requesting the single-step execution mode from MICE.

**Step*** Step timeout. The maximum time (BTUs) to wait for data about a single step.

**Type*** Type timeout. The maximum time (BTUs) to wait when opening a path to MICE and requesting target identification.

**DLd*** Download timeout. The maximum time (BTUs) to spend on downloading a domain for emulation in MICE. The timeout value includes all successive retries.

**DLd#** Download retries. How many retries to make after an unsuccessful download of a domain to the DOMINO controller.

**EscDel** ESCape delay. The time to wait (BTUs) between receiving the results of a requested function and sending ESCape to acknowledge MICE.

### 3.2.12 LIST-MICE-PARAMETERS

**Purpose** List the parameters that determine the communication behaviour between DOMINO Monitor and MICE-II.

## 3.3 Execution commands

The commands in this category are used for loading, starting and stopping execution of an application.

### 3.3.1 SOFT-RESET

**Purpose** This command will perform a software reset on the target. The target enters **aborted** state. The reset is performed by sending a specific command to the target, which means that the effect of this command depends on whether the target is running and able to receive the SOFT-RESET command or not. This command is only available when using ASYL or SERVER as the current path.

## 3.3.2 HARD-RESET

HARD-RESET

**Purpose** This command performs a hardware reset on the target. The target state becomes **aborted.** Both the processor and the device hardware on the target will be put into an initial state. This command is particularly useful after a software crash in the controller. This command is only available when using MICE or SERVER as path.

**NOTE!** Even if OPEN-PATH to a controller does not work, it is possible to send a HARD-RESET to that controller. After HARD-RESET, the user should wait until the selftest has terminated (5-15 seconds depending on memory, SCSI and controller type).

## 3.3.3 STOP-TARGET

**Purpose** The execution of the current application stops, and the target state becomes **stopped**.

If the application, for example goes into an endless loop, and is outputting something on the terminal, user commands will still be received by the DOMINO Monitor. Only echo from what the user types, and output from the Monitor may disappear between the application output. In this case both the DOMINO Monitor and the application may be temporarily stopped by XON/XOFF. CTRL+S halts the program, while CTRL+Q resumes execution.

### 3.3.4 PLACE-DOMAIN

PLACE-DOMAIN <Domain>

**Purpose** The domain is placed in the target's memory and made ready for execution. The program counter is set to the start address of the domain. The target state must be **stopped** or **aborted** before this command is given.

**Domain** Name of the domain. It may be preceded by a SINTRAN user-area in parantheses. COSMOS Remote File Access is also supported, so the complete domain specification becomes:
**System(Remote-user(Password)).(User)Domain**
Both old and new domain format are supported.

Example

```
DM(SERVER): SOFT-RESET↲
DM(SERVER): STOP-TARGET↲
DM(SERVER): PLACE MY-APPLICATION↲

Placing (PACK-ONE:DOMINO-USER)MY-APPLICATION:DSEG
    1042000B is current address.
     243306B bytes transmitted.
Placing (PACK-ONE:DOMINO-USER)MY-APPLICATION:PSEG
    410000B is current address.
     10420B bytes transmitted.

% The application is now ready to be started
```

## 3.3.5 DOWN-LOAD

DOWN-LOAD <File to download>

**Purpose**

Down-load a file from the SINTRAN file system into the controller.

**File to download**

The file must contain Motorola S-record format. S7, S8, or S9 records will modify the PC-register according to the given start address when the RUN command is used.

The Object Converter can be used for making S-records from :NRF format. It can save some time during down-loading, as program variables with no initial values (for example stacks, heaps) are not loaded. On the other hand, S-records are in ASCII format which must be converted into binary format during loading. Debug information is not supported by the Object Converter.

**Example**

```
% use Object Converter to make Motorola format of domain

DM(SERVER): SOFT-RESET↵
DM(SERVER): STOP-TARGET↵
DM(SERVER): DOWN-LOAD MY-APPLICATION:MOBJ↵

   300 is the current record number
Downloading finished.    318 records transmitted.

% The application is now ready to be started by RUN command
```

## 3.3.6 GO

GO (<Address>)

**Purpose** Start execution of a program from <Address>.

**Address** The <Address> is loaded into the PC-register and execution started. If no <Address> is given, execution starts from current value of the PC-register.

**Example**

```
% Assumes that domain in Motorola format is loaded. Its S8
% record is S804020204F3 giving start address 020204H.

DM(SERVER): GO 020204H↲
```

## 3.3.7 RUN

**Purpose** The current domain in the target is started at its start address (main entry). This command requires that a domain has already been loaded by use of either **PLACE-DOMAIN** or the **DOWN-LOAD** command.

## 3.3.8 ATTACH-DOMAIN

ATTACH-DOMAIN <Domain>

**Purpose**

This command is used when a domain is already placed in the target's memory and has been aborted during execution. The command allows investigation of the aborted domain.

**Domain**

The name of a domain in the specified description file. The <Domain> may be prefixed with COSMOS RFA notation:
**System(Remote-user(Password)).(User)Domain**

## 3.4 Macro commands

Macros provide a convenient mechanism for executing the same set of commands repeatedly. As macros may have parameters, they can be regarded as user-defined commands.

Macros are particularly useful for programs requiring certain initialization commands to be given before execution starts, or for executing a set of debug commands.

Each user may in fact build his own set of macros from:

- DOMINO Monitor commands
- SINTRAN commands
- other macros

Macros may be saved permanently in files, or they may just be temporary, vanishing when the DOMINO Monitor is left.

## 3.4.1 DEFINE-MACRO

DEFINE-MACRO <Macro name>
<Macro body> END-MACRO

**Purpose** Compose a new macro from the basic commands or other macros.

Macros defined by this command are temporary. Permanent macros may be prepared by an editor on a file. The DOMINO Monitor expects file type **:MACR**. The number of temporary macros that may be defined are only limited by internal storage (heap) reserved for macros.

**Macro name** The name of the new macro. It can consist of any number of visible characters except space or comma.

**Macro body** Every line following the DEFINE-MACRO command is taken as the macro body until the **END-MACRO** is encountered. It must be written on the beginning of a new line. It can be abbreviated to just **E**. The macro contents will not be checked before execution.

**Parameters** It is possible to define **formal parameters** within the macro body. They are replaced by **actual parameters** when the macro is called. A parameter is defined by

**PARAMETER** <Parameter name> <Default value> <Prompting text>

**PARAMETER** is a keyword that cannot be abbreviated or used for other purposes. If spaces or commas are part of any of the parameter's parameters, they must be enclosed in single quotes ('). Quotes are permitted but not required otherwise.

The first actual parameter supplied in the macro call line replaces all occurrences of the **<Parameter name>** used in the first PARAMETER definition. The second actual parameter replaces **<Parameter name>** used in the next PARAMETER definition, and so on. Excessive parameters are ignored.

**Default value**

If the actual parameter is empty, the default value is used when expanding the macro. Parameters without default are replaced with an empty string when not specified.

**Prompting text**

When a macro is executed, all parameters are prompted for. That means successive parameters cannot be specified on the same line.

**Parameter scope**

Parameter declarations are legal anywhere in the macro body. This means that parameters can be declared after some macro statements. The scope of the declaration is from the declaration point to the end of the macro (provided that it is not redeclared).

## 3.4.2 EXECUTE-MACRO

EXECUTE-MACRO <Macro name>, (<Parameters> ...)

**Purpose**

The macro with the specified name is processed. Formal parameters are substituted with actual parameters.

**Macro name**

The name of an existing (temporary or permanent) macro.

**Parameters**

Actual parameters to replace the formal parameters in the macro. Each parameter must be specified on a separate line. The parameter may contain any character except space or comma.

The words EXECUTE-MACRO can often be left out. The **search strategy** used for looking up a command or macro is as follows:

- search through list of DOMINO Monitor commands. If a match is found, the corresponding command is processed.
- search through list of temporary macros. If any matching macro is found, it is processed.
- test for permanent macro. If a file matches the specified string (default file type :MACR), it is taken to be a permanent macro and processed. The file system will ensure that, if a file with the specified name is not found under the current user, user SYSTEM's default directory is searched.
- If not yet found, it is assumed not to exist.

Note that the extension **(directory:user)** cannot be put in front of a file name specification, as it is taken to be a path specification. Consequently a macro must either reside on the current user-area or on user SYSTEM. File type extension may be used to overrule the default **:MACR**.

Temporary macros may be defined within permanent macros. Such temporary macros will be erased when the processing of the permanent macro is finished. This feature may only be used if the macro is prepared by an editor.

If a macro is given the name of (or a legal abbreviation of) a DOMINO Monitor command, EXECUTE-MACRO may not be left out.

Example

```
DM: DEFINE-MACRO ?↵
@WHO                      % a simple SINTRAN command
END-MACRO↵

DM: ?
===>   768 YOUNG-HACKER
```

Example

```
DM: DEFINE-MACRO START-DOMAIN↵
PARAMETER P1,,'User name ' % Enter user-area for application
NEW P1
OPEN-PATH ASYL ASYL-DISC
USE-MAILBOX ON 1100B       % Use ASYL and mailbox
PARAMETER P2,,'Domain '
PLACE-DOMAIN P2            % Place and start application
RUN
END-MACRO
```

### 3.4.3 RESUME-MACRO

**Purpose** If the DOMINO Monitor is not able to carry out a statement in the body, the macro is aborted. This command makes it possible to force the processing of it to continue. The macro is resumed at the statement following the one where it was interrupted.

Example

```
DM: EXECUTE-MACRO UNRELIABLE↵
DM: OUTPUT-FILE LOG-FILE:SYMB

NO SUCH FILE NAME

CURRENT MACRO ABORTED

DM: RESUME-MACRO↵          % ignore that log-file is missing
DM: PLACE-DOMAIN MY-APPLIC
DM: RUN
....
```

### 3.4.4 ERASE-MACRO

ERASE-MACRO <Macro name>

**Purpose** The temporary macro is erased (deleted). Permanent macros are erased by using the SINTRAN command:

**@DELETE-FILE <Macro name>:MACR**

**Macro name** The name of an existing temporary macro;

## 3.4.5 DUMP-MACRO

DUMP-MACRO <Macro name>

The named temporary macro is written to a file with the same name as the macro. The macro becomes permanent and can at a later time be executed by using the macro name as a command. If the file does not exist, it will be created. The default type of the file is :MACR. The macro name must therefore be an acceptable file name (any combination of letters, digits and hyphens of maximum 16 characters).

**Macro name** The name of an existing temporary macro. If it does not exist, an empty permanent macro is created.

## 3.4.6 LIST-MACRO-NAME

LIST-MACRO-NAMES <Macro names>

**Purpose** The names of the macros with names matching the specified name are listed on the output file. Only temporary macros are listed. Permanent macros are listed by the SINTRAN command:

**@LIST-FILES <Macro name>:MACR,,**

**Macro names** Macro names or abbreviations of names of the macros to be listed. Default is all macros defined.

### 3.4.7 LIST-MACRO-BODY

LIST-MACRO-BODY <Macro name>

**Purpose** The bodies of the macros matching the specified name are listed on the output file. Only temporary macros may be listed. Permanent macros have to be inspected in an editor.

**Macro name** Macro name of macro body to be listed.

## 3.5 Debugging commands

The DOMINO Monitor has several facilities for debugging an application. The basic commands in the DOMINO Monitor allow for hardware-oriented debugging at the assembly level. In addition, a special version of the Symbolic Debugger has been made available in the DOMINO Monitor. It allows inspection of the program by symbolic variables and routine names as used in the source code.

## 3.5.1 DEBUGGER

**Purpose**

Enter the integrated Debugger in the DOMINO Monitor.

The commands of the Debugger are documented fully in the manual Symbolic Debugger User Guide (ND-860158).

The DOMINO Debugger can only be started if a domain has been placed in the controller with the command **PLACE-DOMAIN**, and this domain is the one to be debugged. The Debugger communicates with the target in transparent mode. The target state must be stopped or aborted when entering the Debugger.

In order to use symbolic names, the program must be compiled with the **DEBUG-MODE** option in the compiler turned **ON**. If the DEBUG-MODE option is **OFF**, the DOMINO Debugger may be used, but no symbolic references can be made. All debugger information is stored together with the object code.

It is possible to exit and reenter the Debugger without losing any context (for example for performing other Monitor commands).

```
DM(ASYL): PLACE-DOMAIN MY-APPLICATION↲
....
DM(ASYL): DEBUGGER↲
DOMINO Symbolic Debugger.
PLANC PROGRAM.  MY_MODULE.MY_MAIN.186
   % Main entry at line 186 in source program
$RUN↲

Connecting to target. Break character is: 100B
```

The following gives an overview of the available commands in the DOMINO Debugger.

**ACTIVE-ROUTINES**
List current call hierarchy.

**ALIGN-LISTING**
Adjust line numbers of current program to correspond to an old listing.

**BREAK**
Set break point at one of the items routines, labels or source line numbers. A line number is given relative to the start of the program. The previous break point defined by this command (if any) is reset.

An optional parameter is present. It allows for specification of either **<Count>** or **<Condition>**.

**<count>**, tells how many times program control shall pass the breakpoint before execution halts. The execution halts just before performing any statements of the specified item.

(**<condition>**) is for giving a Boolean expression constructed of constants, variables and the operators ( + - < > >< * / ** = ); which must be true when the breakpoint is reached, for execution to halt.

**BREAK-ADDRESS**
Set break point at program address. This commands resets any previous breakpoint.

**BREAK-RETURN**
Break at return from current routine. Error code is displayed. Note that program execution continues.

| | |
|---|---|
| **CLOSE-HISTOGRAM** | Erase information accumulated in the histogram. |
| **COMPARE-DATA** | Compare data of running program with :DSEG file of source program. Differences are reported. |
| **COMPARE-PROGRAM** | Compare program code with :PSEG file of running program. Differences are reported. |
| **CONTINUE** | Resume execution of program. |
| **DISPLAY** | Display variables in current scope. |
| **EXIT** | Resume DOMINO Monitor. |
| **FIND-SCOPE** | Find scope corresponding to a program address. Returns name of module, routine and line number relative to start of routine. |
| **FORMATS-DISPLAY** | Set format(s) used by DISPLAY command. |
| **FORMATS-LOOK-AT** | Set format(s) used by LOOK-AT commands. |
| **HELP** | List commands and parameters. |
| **INCLUDE-COMMANDS** | Make all permanent macros on a file available. |
| **LOOK-AT-XXXX** | Inspect DATA, PROGRAM, REGISTER or STACK. Subcommands similar to those in the DOMINO Monitor's LOOK-AT. Use HELP within LOOK-AT to get a list of subcommands. |

| | |
|---|---|
| **LOOK-AT-LIST** | Displays records in a single-linked list (linear list). The data structure is identified by a pointer to head of the list, **<start>**, and a pointer within the record to next, **<link>**. The parameter **<count>** gives the number of records to be displayed. Type **CR** to display next record. |
| **MACRO** | Erase, list or build a macro. The macro is listed if no parameters given. It is erased if no body is given as 2. parameter. |
| **PRINT-HISTOGRAM** | List the information accumulated in the histogram on an <Output File>. |
| **RESERVE-TERMINAL** | Reserve an additional, free terminal. The user communication with the Debugger switches to this terminal. Communication with the application still goes via the first terminal. |
| **RESET-BREAKS** | Reset current breakpoints. |
| **RUN** | Start program at specified program address. Execution continues if no address is given. The DOMINO Monitor connects to the target. The break character must be typed to return to the Debugger's command processor. |
| **SCOPE** | Switch observation scope to specified, active module or routine. Default is the current scope. This command does not affect the program execution, only the set of variables that may be inspected. |
| **SET** | Assign value to a variable. The value may be a constant or an expression. The variable may be simple or composite. |

**SET-HISTOGRAM**

Define a program area to logged in the histogram. The histogram gives the percentage of CPU time spent on different program parts. The program area is identified by the parameters **<Start address>** and **<Maximum address>**. The program area is divided into **<interval>** equal partitions, logged individually. The maximum is 64 intervals.

This command can be repeated several times to cover several program fractions in the histogram.

**STEP**

Step through the program instruction-by-instruction. The **<count>** parameter must be -1. The optional parameters (**<low>**) and (**<high>**) specify the program area where the step mode is active. If not given, step mode is used on the entire program. The instruction executed is output. Each **CR** typed causes execution of the next instruction.

**PROGRAM-MAP**

Print a map of a specified module or routine. The following is output: Program area (addresses), entry point, stack demand, variables with type and initial values. This is very useful when doing assembly-related debugging. By giving this command in a mode job, you may obtain a list to be printed on paper.

**USE-HISTOGRAM**

Switch the use of the histogram **ON** or **OFF**. Information is only accumulated in the histogram when this switch is ON. No information is erased before **CLOSE-HISTOGRAM** is given.

**Operators**

The following operators are available in most expressions: + -Shift Addr Mod TypeOf * / ** . (dot). Symbolic names cannot be abbreviated as they have to be unique.

Some examples

```
$BREAK-ADDRESS Addr( <Routine name> ) % same effect as BREAK

$BREAK-ADDRESS Addr( <Line number> )  %     ---- " ----

$DISPLAY <Pointer name>               % inspect pointer

$DISPLAY Ind( <Pointer name> )        % inspect data element
                                      % of pointer

$DISPLAY Addr( <Variable Name> )      % address of variable

Integer : i                           % somewhere in source
13 =: i
...
$DISPLAY ADDR(I)                      % verify I has changed
ADDR(I) = 00000400044B
$COMPARE-DATA
Low: 400044B                          % segment no ' address
Low: 400044B
D 00000400044B: 00B CHANGED TO 015B

$DISPLAY
MaxChar=127  NoBytes=0
Prompt= (00000532364B;0:14) Default= (Nil;0:0)%byte pointers

$SET NoBytes = 125                % SET using constant
$SET NoBytes = MaxChar-2          % SET using simple variable
$SET Default = Prompt             % SET using composite data
```

```
$DISPLAY
MaxChar=127  NoBytes=125
Prompt= (00000532364B;0:14) Default= (Nil;0:0)%byte pointers

$SET Ind(Prompt) = 0                  % clear buffer
$SET Ind(Prompt) = 'Hello'
$DISPLAY Ind(Prompt)
IND(PROMPT)=HELLO

% Suppose you want a histogram from line 20 to 34

$DISPLAY ADDR(20); DISPLAY ADDR(34)
ADDR(20)=0..400036B  ADDR(34)=0..400242B
$SET-HISTOGRAM 400036B 400242B 14
$USE-HISTOGRAM ON
$BREAK 35
$RUN
....                                  % CPU-time per interval
$PRINT-HISTOGRAM                        In this example is
APPLIC.20        0..400036B       5.80  almost all CPU-time
APPLIC.22        0..400060B       0.00  spent outside the
....                                    logged program area.
APPLIC.33        0..400234B       0.00
```

NOTE! LOG-LINES and LOG-CALLS and some others commands are not available, as DOPCOM does not support multiple breakpoints

## 3.5.2 BREAK

BREAK <Address> (<Count> /1/ ) (<Commands>)...

**Purpose**

Set breakpoint at a program address. When the breakpoint is reached, execution terminates and control is passed to the command processor.

It is possible to set new breakpoints as long as DOMINO Monitor has memory space to store information about them. The breakpoints are active until reset by the **RESET-BREAKS** command.

**Address**

The program address where a breakpoint is to be set.

**Count**

How many times the program control shall pass the breakpoint before breaking. The execution stops just prior to executing the instruction at the breakpoint address.

**Commands**

DOMINO Monitor commands to be performed when the breakpoint is reached. Default is none. Maximum 7 commands can be given. It is legal to invoke macros.

After a breakpoint has been reached, program or data locations or the registers may be displayed or modified. The next instruction to be executed is by default the instruction pointed to by the PC-register, but this may be overridden by the **GO** command or the optional <Start address> parameter of the **STEP** command.

**Example**

```
DM(SERVER):BREAK 400136B 1 LOOK-AT-DATA 677160B↵
DM(SERVER):RUN↵

% Execute until breakpoint is detected, application
% terminates or application is aborted due to error.
```

## 3.5.3 TEMPORARY-BREAK

TEMPORARY-BREAK <Address> (<Count> /1/ ),
(<Commands>)...

**Purpose** Similar to **BREAK** except that, when the breakpoint is reached, the breakpoint is reset.

**Address** The program address where the breakpoint is to be set.

**Count** The number of times the program control should pass the breakpoint before breaking. The execution stops just prior to executing the instruction at the breakpoint address.

**Commands** DOMINO Monitor commands to be performed when the breakpoint is reached. Default is none. A maximum of 7 commands can be given.

## 3.5.4 STEP

STEP <Start address> (<Count> /1/ )
(<Commands>)...

**Purpose** Enter single step mode. If no parameter is given, the instruction pointed to by the program counter is disassembled and displayed.

By typing **CR**, the instruction pointed to by the PC-register is executed. **CR** can be repeated several times. Typing anything else causes return to the DOMINO Monitor's command processor.

**Start address** The program address where single-step execution should start. Default is the current value of the program counter.

**Count** The number of times the program control should pass the <start address> before entering single step mode. The execution stops just prior to executing the instruction at this address.

**Commands** DOMINO Monitor commands to be automatically executed in single-step mode. Default is none. A maximum of 7 commands can be given. The commands are executed between each step. The STEP command must not be called again.

Example

```
For i In 1:100 Do  % source program area to be stepped
  $* NOP           % is a wait loop
Endfor

DM(SERVER): STEP↵
     401016B: MOVE.Q  #1,DO          % Initial value of i
     401020B: EXT     DO; EXT.L  DO  % Sign extend DO
     401024B: MOVE.L  DO,677160B     % Save current i
   ┌→401030B: NOP                    % Body of loop
   │ 401032B: MOVE.L  677160B,DO     % Restore current i
   │ 401036B: ADDQ.L  #1,677160B     % ++ i (next valid i)
   │ 401042B: CMPI.L  #144B,DO
   └─401050B: BNE.S   *-20B          % Repeat if i >< 100
```

## 3.5.5 RESET-BREAKS

**Purpose** All breakpoints are removed by using this command.

## 3.5.6 RESET-LAST-BREAK

When a breakpoint is encountered during execution, this breakpoint may be removed and the original instruction restored by executing this command.

## 3.5.7 DEBUG-STATUS

**Purpose** List information about defined breakpoints.

## 3.5.8 SET-SPECIFIC-ACCESS

SET-SPECIFIC-ACCESS <ON/OFF: OFF >

**Purpose** Turn on or off the specific memory access mode used during debugging. If it is **OFF** when the LOOK-AT command is used, the Monitor will prefetch a whole block of data from the DOMINO controller's memory. This happens even when only a single memory location is to be displayed. If it is **ON**, only the unit of information (byte, halfword or word) actually needed at the moment will be fetched.

**ON/OFF** It is a good rule to let the switch be OFF if several locations are to be investigated at the same time in the same memory area, and to let it be ON for sporadic investigation.

## 3.5.9 MAIN-FORMAT

MAIN-FORMAT <Format: /OCTAL/ >

**Purpose** Set the numeric format to be used when displaying numbers. Octal is set as main format when the DOMINO Monitor is entered.

**Format** OCTAL, HEXADECIMAL or DECIMAL or abbreviation of one of these.

## 3.5.10 EXTRA-FORMAT

EXTRA-FORMAT (<Formats>)...

**Purpose** Sets additional, numerical format(s) to be used when displaying numerical values.

**Formats** Any of the formats BYTE, HALFWORD, ASCII, OCTAL, DECIMAL, HEXADECIMAL. The names of the formats can be abbreviated.

If no <Formats> are given, the extra formats are switched off.

## 3.5.11 LOOK-AT-PROGRAM

LOOK-AT-PROGRAM <Address /0/ >

**Purpose** Display and modify program data. Several subcommands are available.

**Address** The memory address from where inspection should start.

## 3.5.12 LOOK-AT-STACK

**Purpose**

The current local data field is displayed. This is the memory area pointed to by the current A6-register (used as stack pointer), and contains routine call information, such as address to local data field and return address to calling routine.

Several subcommands are available. The subcommands **PREVIOUS** and **NEXT** are only related to **LOOK-AT-STACK.**

**Subcommand PREVIOUS**

Display the previous local data field (for example the local data field of the calling routine). This command may be repeated until reaching the local data field of the main program, which has the lowest stack frame.

**Subcommand NEXT**

Display the next local data field (for example the local data field of the procedure called by the current one). It is only valid to do this after PREVIOUS. It is not possible to move beyond the data field of the routine currently being executed (the uppermost stack frame) of the current call hierarchy.

**Stack format**

The stack frame format for ordinary routines (valid from H-version of the PLANC-MC compiler) is as follows:

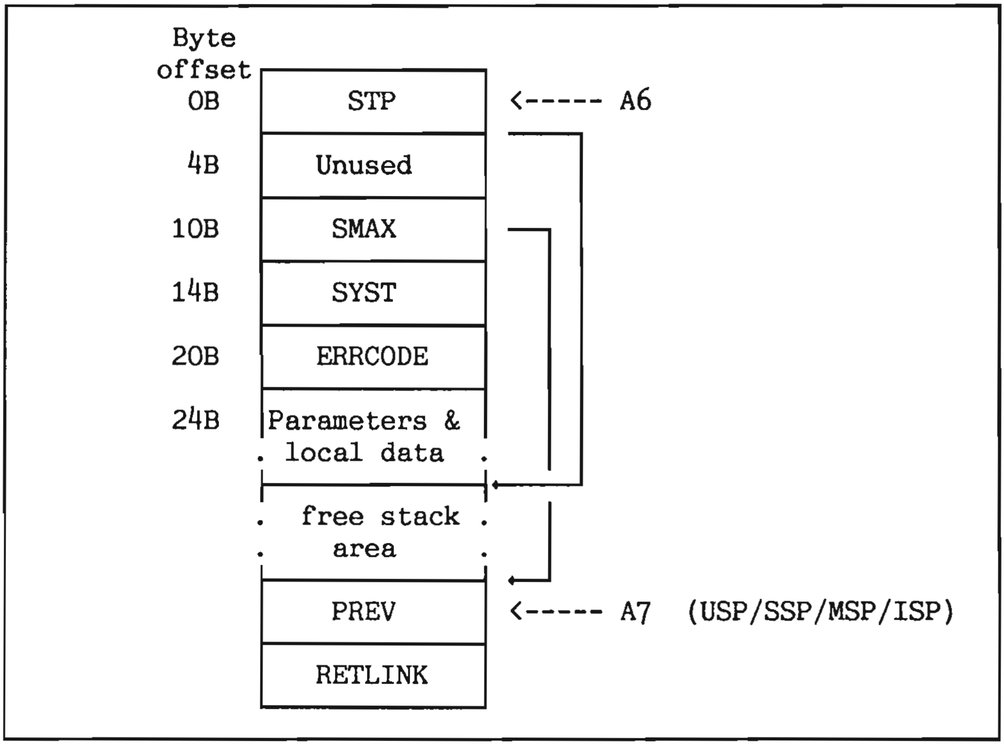



*Figure 8. PLANC-MC ordinary stack frame*

**STP -**
**STack Pointer**
Points to the first free location of the stack. The stack grows both upwards and downwards.

**Unused**
Reserved word for future extension.

**SMAX -**
**Stack Max**
Points to the top of the free stack. This is the same as the A7-register for the current stack. The variable is needed as there may be several stacks in use. The value of A7-register changes after each stack initialization (**Inistack**).

**SYST**
Reserved word for PLANC runtime SYSTem.

**ERRCODE**
The value of ERRCODE of current routine.

**Parameters** Actual parameters are placed on the stack in the same order as they are declared. A routine with in-value or out-value is passed in another way: Simple variables and constants not exceeding 32 bits are passed via the D0-register. All other variables and constants as pointer to the actual parameter are passed via the A0-register.

**PREV** The previous value of the A6-register. The previous value of A7-register is A7 - 2 words. Both registers are restored with previous values at routine termination.

**RETLINK** The return address of the calling routine. If the routine terminates normally (not ERRETURN), this address is incremented by two (bytes) when returning (also called skip return).

**ErReturn** If a routine makes error return (**ErReturn**), a jump is made to the PLANC runtime routine #XRET. The address of #XRET is always in the A5-register. The #XRET routine performs error return to the previous level. The current stack frame is popped on the stack. The D0-register keeps the ERRCODE value.

The instruction following a routine call (content of address RETLINK) holds either a subroutine call to the local exception handler, or a new jump to #XRET if no local handler is defined (**On RoutineError Do ... Endon**). In this way control passes to the next higher routine in the call hierarchy. All routines at lower call levels than the one having the exception handler are terminated.

**Special routine** The **Special** routine cannot have parameters, except for the in-value and out-value. No local stack is initiated for the routine when called. The routine has to do this itself if any local data is to be used.

**Native routine**

The **Native** routine is well suited for use by exception handlers. It can have in-value and out-value (which an interrupt routine usually does not need), but no formal parameters. The local stack is initiated when activated, allowing for local variables. A slightly different **stack frame** is used:

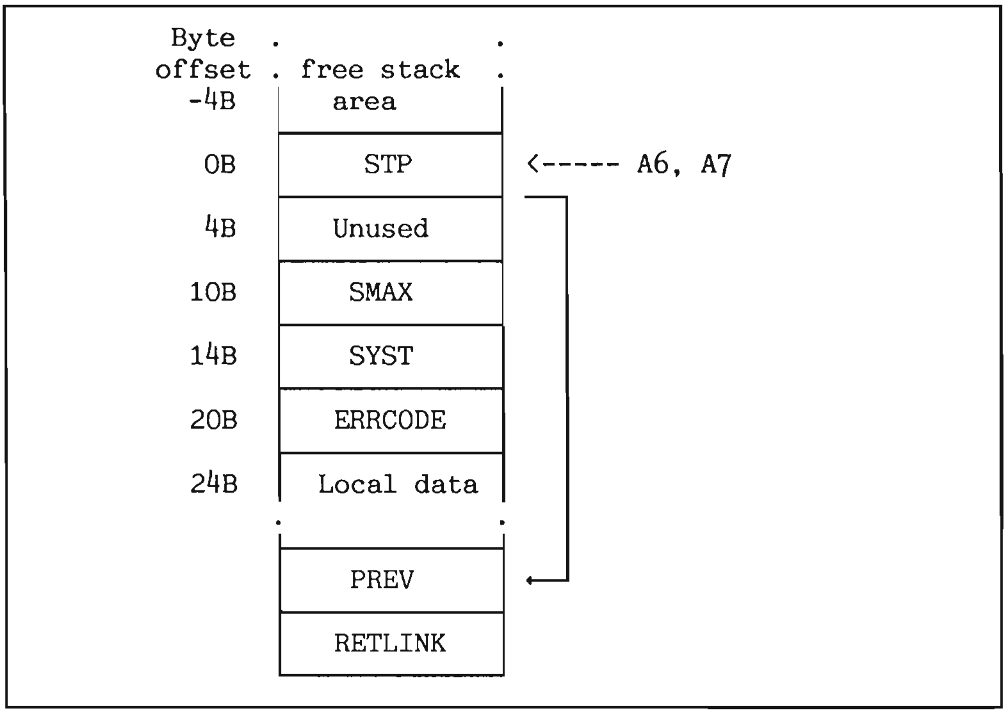



*Figure 9. PLANC-MC native stack frame*

**STP - STack Pointer**

STP points to 1. free location after local data (PREV). The stack grows only from high to low memory addresses. This is similar to how the CPU uses the stack.

**PREV** The previous value of the A6-register. The previous value of the A7-register is STP - 2 words. Both registers are restored with previous values on termination of the routine.

**RETLINK** Return address to calling routine.

**Native restrictions**

- It is not possible to make an error return from a native routine.
- An ordinary routine can call a native routine but not the opposite way around.
- A native routine can call other native routines.
- A native routine can have ordinary routines as inner routines.
- No inner PLANC routine can be called recursively.

## 3.5.13 LOOK-AT-RELATIVE

LOOK-AT-RELATIVE <Relative to> /A6/

**Purpose**

Start listing of contents in memory relative to either the contents of a register or absolute address. Both absolute and relative addresses are displayed. Several subcommands are present.

**Relative to**

Any register or a numeric address. Default is A6-register (PLANC stack pointer).

## 3.5.14 LOOK-AT-REGISTER

LOOK-AT-REGISTER <Name> /PC/

**Name**

The name of one of the registers. The specified register is displayed in current main format. If **CR** is typed, the next register in the sequence is displayed. Several subcommands are present.

The registers are: PC, D0:D7, A0:A6, USP, SR, SSP, ISP, MSP, VBR, SFC, DFC, CACR, CAAR. The A7-register is at any time one of the stack pointer registers: USP (User SP), SSP (Supervisor SP), MSP (Master SP) ISP (Interrupt SP). Only MC68020 has the registers MSP, ISP, CACR and CAAR.

## 3.5.15 LOOK-AT subcommands

This set of subcommands can be used to inspect several items in succession, change displayed format, change items to be inspected, modify contents of registers or memory.

**EXIT**

Return to DOMINO Monitor command processor. In addition to the command EXIT, both a full stop (.) or a semicolon (;) terminate the LOOK-AT subcommands.

**HELP**

All subcommands matching <Command> are output together with their parameters.

**PERMIT-DEPOSIT**

In order to avoid unintended modification of the memory or a register, the command **PERMIT-DEPOSIT** must be typed before the depositing of a new value can take place. An exception is when the CODE command is used.

**↵**

Carriage return causes display of the next item (register, instruction or memory location).

**<A>, <N> / <file> ↵**

Dump <N> bytes starting at address <A>. <A> may also be a register name. If <file> is given, the dump is written into this file.

Any of the parameters may be omitted, causing the default values to be used. Default value for <A> is the current address inspected, default value for <N> is the number of bytes within the current format, while omitting <File> will cause the output to be written to the standard output file (for example, terminal).

**<n> ↵**

Modifications of memory or registers are made by typing the new value <n> followed by CR. <n> is deposited into the current memory address or register inspected. The current address can be altered by typing <A> / **CR**.

```
Example

DM: LOOK-AT-REGISTER PC↵
PC:   700000B   PERMIT-DEPOSIT↵
PC:   700000B 400000B ↵      % PC is changed
                   P/
PC:   400000B↵               % Verify change
```

**'<string>' ↵** The memory can be modified by an ASCII string by enclosing the <string> in single quotes. Two successive quotes are interpreted as one single quote (for example ''' becomes ').

**CODE <Instruction>** Assemble symbolic assembler instruction and deposit into memory. Several instructions can be given simultaneously by separating each with a semicolon (;). The instruction(s) will be assembled and stored, starting at the current location. Program memory may also be modified numerically by first typing **BYTE**, and thereafter modifying bytes in the main format (see the MAIN-FORMAT command on page 80).

```
Example

% Removing a test by patching

DM(SERVER): LOOK-AT-PROGRAM 400160B↵
     400160B: BNE.B  *-22B   CODE↵
     Instruction: NOP↵
     400160B: NOP
```

**BREAK** Sets a breakpoint in the current address. The command is similar to the BREAK command. Parameters are <Count> and <Commands>.

Example

```
DM(SERVER): LOOK-AT-PROGRAM 400160B↲
     400160B: NOP BREAK↲
     400160B: BKPT      #7 BREAK

     % The original instruction is copied to the breakpoint
     % table inside OPCOM, before being replaced by the BKPT
     % instruction
```

**TEMPORARY-BREAK** Sets a temporary breakpoint at a current address. The command is similar to the TEMPORARY-BREAK command in the DOMINO Monitor. Parameters are <Count> and <Break>.

**Change format** When displaying memory it is possible to use BYTE, HALFWORD (16 bits), or WORD (32 bits) as main display format. DISASSEMBLE can be used for getting symbolic assembler instructions (for example when moving into a memory area containing instructions when using LOOK-AT-DATA).

Additional display formats may be obtained by typing **EXTRA-FORMAT** followed by a list of formats. This command is similar to the global EXTRA-FORMAT command, except that the extra formats are only valid within LOOK-AT.

**COMPUTE <Expression>** Evaluates and displays the result of an arithmetic expression. It is displayed in all numeric formats. The command is similar to the global COMPUTE command.

**ABSOLUTE <Address>**

Displays an item from an absolute address. Addresses are otherwise taken as relative addresses.

**Change mode**

In a LOOK-AT command, it is possible to change to one of the other LOOK-AT commands by typing one of the subcommands below. This is equivalent to EXITing from LOOK-AT and typing another LOOK-AT command. This feature saves some typing work. The modes available are:

- DATA <Address>
- PROGRAM <Address>
- REGISTER <Name>
- STACK
- RELATIVE <To>

## 3.6 DOMINOS process monitoring

These commands are only relevant when running applications under control of DOMINOS.

## 3.6.1 PROCESS-STATUS

PROCESS-STATUS <Process name>

**Purpose**

Print status of the processes matching <Process name> on the output file.

**Process name**

Any abbreviation for a process name. Default is all processes.

The information given for each process is:

- Process name
- Process state (DORMANT, BLOCKED, READY or RUNNING)
- Process priority
- Event buffer (events set but not yet read by the application)
- Program Counter
- CPU time used, measured in units of 5 ms.

If the parameter <Process name> matches exactly the name of an active process (not DORMANT), the process context is displayed:

- Data registers D0...D7
- Address registers A0...A6
- User Stack Pointer USP
- Status Register SR

Note that the register contents is undefined for a DORMANT process.

If the process is scheduled after a round-robin strategy (among processes with equal priority), **time limit** is displayed. That is, how many time units to use before being moved backwards in the ready queue. 0 is interpreted as 2**32 time units (244 days 15 hours).

If the processor is running in supervisor mode at the moment (for example, DOMINOS service is being executed for a process or an exception handler is active after an interrupt), **SSP** (Supervisor Stack Pointer) is displayed.

If the process is **BLOCKED** when waiting for event(s) to occur, the **event mask** is displayed.

The **READY** queue of DOMINOS is displayed, showing the processes ready to run by name in the order they will be assigned to the CPU. The first process is the currently executing one.

Example

DM(SERVER): PROCESS-STATUS↲

| process | state | prio | event buffer | p-counter | time used |
|---|---|---|---|---|---|
| PR01 | blocked | 1 | 0 | 0 | 200 |
| PR02 | dormant | 6 | 0 | 20 | 90 |

## 3.6.2 LIST-TIME-QUEUE

LIST-TIME-QUEUE (<Interval /1/ >)

**Purpose** List once or periodically all entries in the time queue.

**Interval** Time in seconds between each report.

Each entry (if valid) contains the following information:

- The name of the process to receive the time scheduled event(s) when the delay time expires.
- The event(s) to be set.
- The remaining delay time (in 5 msec units).
- The interval time (in 5 msec units). The delay time to be used together with periodic scheduled events. 0 means no periodic scheduled events.

There may be entries in the queue which are no longer valid, since the service request has been cancelled. In this case the word "VOID" is displayed.

**Example**

```
DM(SERVER): LIST-TIME-QUEUE↵
process          events           delay         interval
PR01                  2             250                0
```

## 3.7 DOMINO controller commands

These commands are related to the hardware environment in the controller.

### 3.7.1 SET-PROTECTION

SET-PROTECTION <From address> <To address> <Supervisor mode> <User mode>

**Purpose**

The DOMINO controller has a flexible memory protection system. The memory protection can be changed dynamically while running programs. The microprocessor's user and supervisor mode of operation can be given separate access rights for the same memory area. The command can be repeated to protect several areas.

When the controller starts, a default protection setting is made. This is modified if DOMINOS is loaded and started.

**Address**

The <From address> and <To address> give the memory area to be protected. Seen from hardware the local memory is divided into segments of 1024 bytes each which can be protected individually.

**Supervisor mode**

Access rights for area when the microprocessor runs in supervisor mode.

**User mode**

Access rights for area when the microprocessor runs in user mode.

The basic legal access rights are: Fetch, Read-Write, Read-Only, No-Access. Fetch means that the contents of the memory area can be executed as instructions. Fetch should normally not be used together with Read and Write. Two PIOC-compatible modes are supported instead.

| Supervisor | User |
|---|---|
| Any-Access<br>Read-Fetch | Any-Access<br>Read-Fetch |

*Table 2. PIOC-compatible memory protection*

There are four combinations of user and supervisor mode that are **not** allowed.

| Supervisor | User |
|---|---|
| Fetch | Read-Write |
| Fetch | Read-Only |
| Read-Write | Fetch |
| Read-Only | Fetch |

*Table 3. Memory protection not allowed*

## 3.7.2 USE-PROTECTION

USE-PROTECTION <ON/OFF: /OFF/ >

**Purpose** Switches the entire memory protection system **ON** or **OFF**. The memory protection is switched on during controller initialization, and after having started DOMINOS.

## 3.7.3 LIST-PROTECTION

Lists the memory-protected areas.

## 3.7.4 USE-CACHE

USE-CACHE <ON/OFF: /OFF/ >

| | |
|---|---|
| **Purpose** | Switches the use of the MC68020's cache **ON** or **OFF**. The use of the cache is switched on during controller initialization. |

## 3.7.5 TARGET-IDENTIFICATION

| | |
|---|---|
| **Purpose** | Gives mainly static information about the target by displaying the following: |
| **CPU Type** | MC68000, MC68010, MC68012 or MC68020. |
| **PROM 1 version** | Version and revision number of DOPCOM. |
| **PROM 2 version** | Version of optional (device-dependent) PROM. |
| **RAM size** | Size of controllers local memory in bytes and pages. |
| **Standard tests** | Indicates whether the self-tests have been correctly executed or not. |
| **Device tests** | Indicates whether the optional (device-dependent) tests have been correctly executed or not. |
| **Trace module** | Indicates whether trace module is present or not. |
| **Trace PROM version** | Version of the trace module if present. |

## 3.7.6 TARGET-STATUS

**Purpose** Gives information about the current state of the target by displaying:

**Controller state** The current state of the application program can be:

- running : Application is running normally in the controller.
- stopped : Application is suspended or has properly terminated.
- aborted : Application has been stopped due to serious error.
- Waitcount: A power drop has moved the controller to this state. "GO" will continue the application.

The following three states are internally used, and should not be visible with the TARGET-STATUS command:

- prestep
- stepping
- PFoccured

**Cache mode** Indicates whether the MC68020's cache is being used or not.

## 3.7.7 SCOPE-LOOP

SCOPE-LOOP <Loop type> <Data type> <Address 1> <Address 2> (<Pattern 1> <Pattern 2>)

**Purpose** Defines and starts a short program loop. This is intended specifically for hardware debugging via oscilloscope, logic analyzer or tracer. The loop consists of two memory accesses followed by optional compare of data.

**Loop type** Read, Read-Compare, Write, Write-Read, Write-Read-Compare

**Data type** Byte, halfword or word (32 bits).

**Address 1 and 2** The memory addresses to be accessed as defined by **Loop type**

**Pattern 1 and 2** If the loop type includes **write** of data, Pattern 1 is written into Address 1, and similarly for the second pair of parameters. If the loop type includes **Compare**, the patterns are used as "the data expected".

# Chapter 4 Applications in DOMINO

## 4.1 Getting started

In this section, you find a small example of how an application is written, loaded by the DOMINO Configurator and run in the controller. It shows what is required of a **minimum** DOMINOS configuration with one trivial application process. The application runs in an endless loop and prints a message on the terminal every second.

The routine ERRCHECK in the following example is to be imported in many of the subsequent examples for handling the return status of DOMINO services.

Source code for DOMINOS-TEST:PLNC

```
$Include DOMI-DEFINES:DEFS
Module Common
   Export ErrCheck, WaitSeconds
   Routine Integer, Void : Errcheck % display error code
      If @ >< PIOK Then
         Output(1,'A','$Error occurred, errcode = ')
         Output(1,'O',@), Output(1,'A','B')
      Endif
   Endroutine

   PITUniWaitEv : WaitRec := (PIEvsel,(0,1),200,1,0)

   Routine Integer, Void : WaitSeconds
      % block (suspend) calling process @ no. of seconds
      @ * 200 =: WaitRec.PITimeOut
      Addr (WaitRec) PIRUniWaitEv  ErrCheck
   Endroutine
Endmodule

Module PRO2
   Export PRO2
   $Include DOMI-APPL-IE:IMPT
   Import (Routine Integer, Void : WaitSeconds)
   Integer Array : S (0:1023)
   Program : PRO2
      IniStack S
      Do % forever
         Output(1,'A','$PRO2 running')
         1 WaitSeconds
      Enddo
   Endroutine
Endmodule

Module Auto_Start
   Export Auto_Start
   $Include DOMI-APPL-IE:IMPT
   Import (Routine Integer, Void : ErrCheck)
   Import (Program : PRO2 )
   Integer Array : S (0:1023)
   Constant Prior = 5
   PITCreate : CreRec := (0,'PRO2',PIABegin+Prior,Addr(PRO2))
   Program : Auto_Start
      IniStack S
      Output(1,'A','$Creating process PRO2')
      Addr (CreRec) PIRCreate =: Errcode
      Errcode Errcheck
   Endroutine
Endmodule
```

- DOMINOS expects that the process **Auto_Start** is exported, and that it takes care of starting other processes. Auto_Start is automatically given the name PRO1 by DOMINOS.

- Auto_Start runs with priority 1, which is the lowest possible one. Each time a process is created and ready to run ("create and go"), the queue of runnable processes is scheduled. If the new process has priority > 1, it will immediately gain access to the CPU. Thus, Auto_Start does not get the chance to finish its work first. This may be avoided by giving Auto_Start higher priority than the processes it is going to start.

- The PLANC compiler automatically includes a system call (MONO) to terminate the process at the end of the program. Therefore, it is usually not necessary to terminate the process by an explicit call.

- There must be at least one separate module for each application process. Each process must have a **main program** and its own **stack**.

- The include file with imported routines must be included in each module. The DOMINOS data types need only be included at the outermost level. These are PLANC restrictions.

- Output can only be sent to the user's terminal when the DOMINO Monitor is run, or when a service terminal is attached to the controller.

- Instructions for using DOMINOS services and for building records are given in the remainder of this section.

```
Compiling DOMINOS-TEST:PLNC

@ND-500-MONITOR↲
ND-500 MONITOR Version H00
N500: PLANC-MC68↲
- MC-68020 Planc Compiler - May 15, 1987
*DEBUG-MODE ON↲
*COMPILE DOMINOS-TEST:PLNC,,DOMINOS-TEST:NRF↲
    335 Lines compiled.   No diagnostics.
*EXIT↲
```

- It is convenient to compile the application together with debug information, as there will be no need for recompilation before debugging when unexpected results occur. The debug information does not slow down the execution of the application. The only disadvantage is that the :NRF and :LINK files become a little larger.

- DOMINOS has also been compiled together with debug information, so all code in the DOMINO controller (except OPCOM) can be referred to by symbols in the Symbolic Debugger.

# Chapter 5 DOMINOS

DOMINOS is an operating system common to all the DOMINO controllers. The functions offered by DOMINOS are described in this chapter.

DOMINOS is backwards compatible with PIOCOS on source level from the programmer's point of view, except for a new **naming convention** used in include files. DOMINOS is **not** an updated PIOCOS but a completely new implementation, incorporating a similar architecture but more efficient **algorithms** and **tools**.

## 5.1 DOMINOS configuration

From version BOO of DOMINOS the DOMI-GENERATOR is replaced by the so called Configurator. Both programs implement a similar solution: Depending on user defined input a mode file is produced and started. The input is now no longer the answer to a lot of questions but the configurator resembles a compiler which compiles a small "high-level programming language" into a mode file.

**Configuration language syntax**

The diagrams show the current syntax version of the language. Words in lower case are reserved keywords, upper case refers to a different syntax diagram and <....> refers to user selectable file names, routine names and so on. Note the use of strings!

The diagram on page 107 shows the overall structure of a configuration program. It starts always with the definition of the target hardware. For the time being only two device types are possible: VENUSGLUE or MPMSTDDOMINO (= MPM based STandarD DOMINO).

The next statement is optional and allows to specify a string which - executed as a SINTRAN command (the "@" must not be included into the string) - activates the linkage-loader to be used. If this statement is used, the programmer has to be aware that one of the next versions of the configurator will assume the use of the new ND-LINKER. A change in the configuration program will in this case become necessary.

The DOMAIN statement is NOT optional, however the WITH part can be omitted with the result that the same name is used for domain and segment. SINTRAN filename syntax can be used for both domain and segment.

THE PLACE statement defines the base address of DOMINOS, default is 400000B, which is the start of the area reserved by the OPCOM module for applications.

With the optional BUFFER statement the system buffer pool of DOMINOS can be increased by the amount of <value>. No decrement is possible.

If the default name of the :NRF file containing DOMINOS can not be used for example because a SINTRAN user area will be included, the SYSTEM statement must be used.

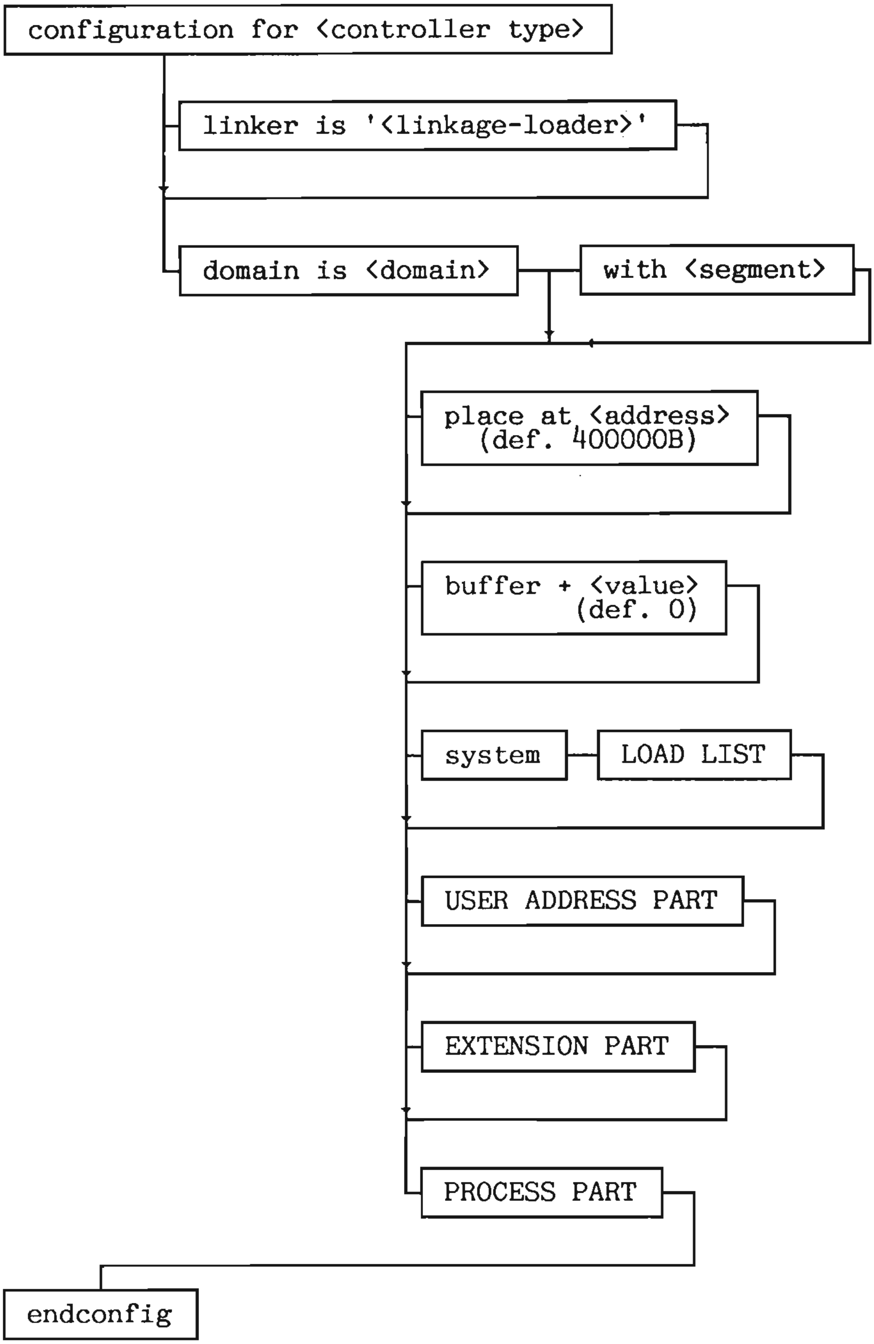



*Figure 10. Structure of a DOMINOS configuration program*

**USER ADDRESS**

The User address part is a complex and optional statement which starts with the reserved word USERADDR. Here specific user entries/addresses can be specified. Each of the currently three entry/address specifications is optional.

- The first - starting with the reserved word ENTRY allows to specify a different user entry. By default the user entry is a PLANC PROGRAM : AUTO_START . (This possibility is new in the configurator, no predecessor in the GENERATOR.)

- The second specification - starting with "DATA AT" allows to specify where the user data should be located. Using this is necessary if user program and data overlaps or if the user wants to save memory space.

- The last specification allows to change the address range where processes (in MC68K user mode) have read/write access (in addition to the user memory data area) on the MC68K bus if at least one of the processes is created with the "system" bit set. (refer PIRCreate service in DOMINOS). The default is the range reserved for the device part on DOMINO controllers.

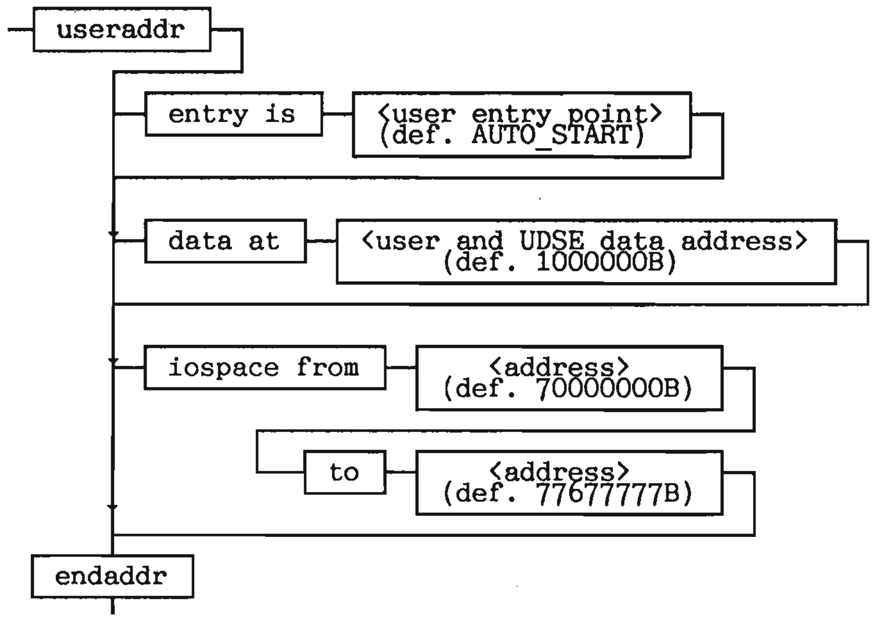



*Figure 11. DOMINOS configuration. USER ADDRESS PART*

**EXTENSION**

The optional complex EXTENSION statement, specifies in the LOAD LIST all files which contain user defined system extensions (UDSE). (Even files which contain exception handlers must be included into this list, although the exception handlers are not specified here but at runtime with the service PIRCREATE.)

Four branches inside the EXTENSION statement allow to specify up to eight (2 x 4) process mangement extensions (PME) in the BEGIN/END ACTION branches, and up to eight user defined services (UDS) in the CALL branch.

The EXTENSION statement must be closed with ENDEXT. (Code and data of UDSE is now located in DOMINO memory, that it can be called/accessed in supervisor mode as well as user mode. The result is that it is no longer necessary to have two copies of the same part in memory, one in the UDSE area and one in the process area.

**Libraries must be loaded twice**

Libraries MUST be loaded with the UDSE first, and then once again with the processes. In the second load only those modules which are accessed by processes, and not by UDSE are placed in memory. To drop loading the library first with UDSE will not result in undefined references BUT IN MEMORY PROTECTION VIOLATION when the UDSE calls the library routines which then reside in process memory!) For a detailed description of user defined system extensions read the section DOMINOS FOR ADVANCED PROGRAMMERS on page 151

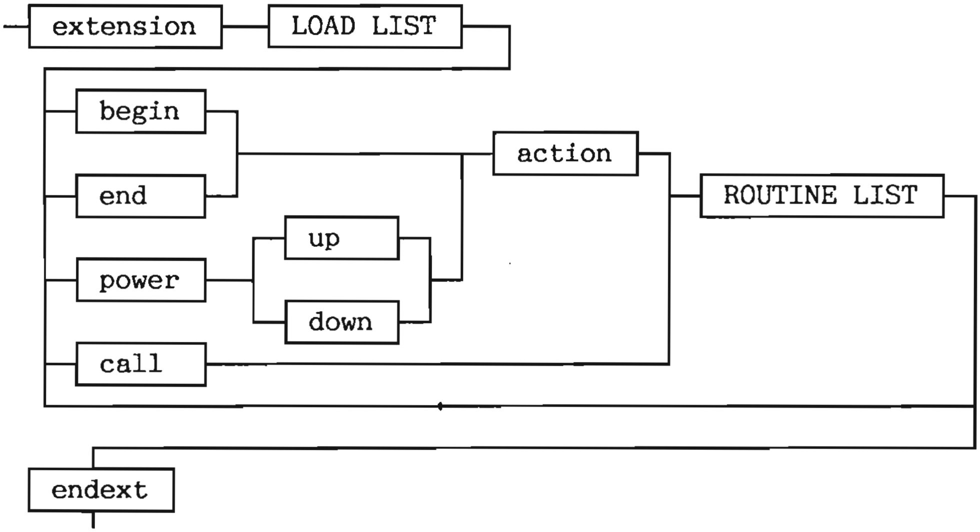



*Figure 12. DOMINOS configuration. EXTENSION PART*

**PROCESS PART** The PROCESS statement contains for the time being only a LOAD LIST specifying all files which contain process code. Concerning libraries refer to the EXTENSION statement above. The PROCESS statement is NOT optional.

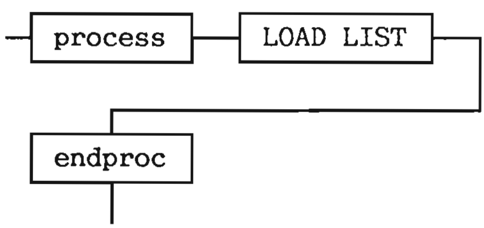



*Figure 13. DOMINOS configuration. PROCESS PART*

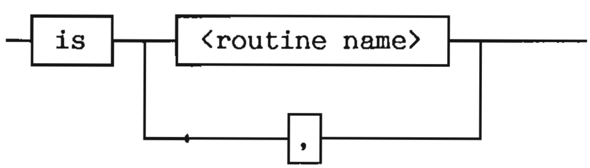



*Figure 14. DOMINOS configuration. ROUTINE LIST*

**LOAD LIST** Each file name in the LOAD LIST command is converted to one load command for the linkage loader. Between the load commands the user can insert anything he wants by using the INSERT command.

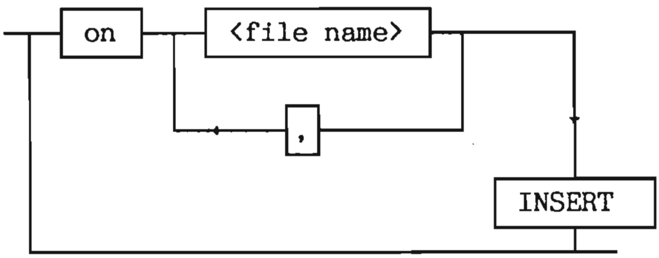



*Figure 15. DOMINOS configuration. LOAD LIST*

**INSERT** The optional INSERT statement is used in connection with LOAD LIST and the SKIP TO statement (refer below). Its purpose is to enable the user to include special commands into the generated mode file to tailor it to his own needs. Each string in the INSERT statement is placed on a new line in the mode file.

> NOTE! The use of INSERT statements can result in incompatibility when larger changes in the generated mode file structure are introduced with a new version of the configurator.

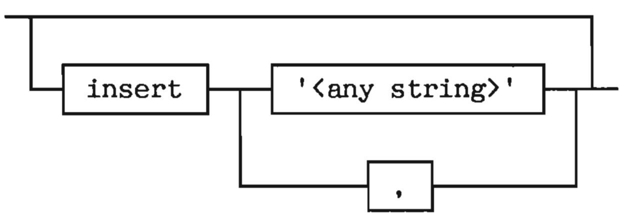



*Figure 16. DOMINOS configuration. INSERT*

**SKIP** The following diagram illustrates the SKIP statement.

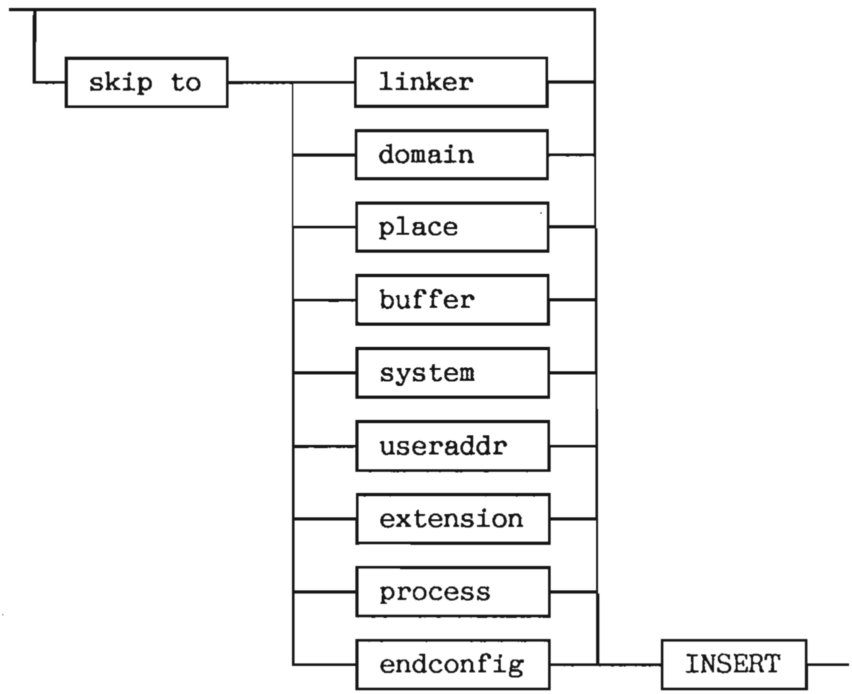



*Figure 17. DOMINOS configuration. SKIP*

The SKIP statement can be inserted into the configuration program such that its label (one of the keywords LINKER...ENDCONFIG, see diagram) appears before the statement which starts with the same keyword appears. The effect is that the compiler generates the mode file up to that statement (unless there is a non-default statement in between). This becomes important together with the use of the INSERT:

An example

```
CONFIGURATION FOR VENUSGLUE

  DOMAIN IS "DOMINOS-C"

  SKIP TO BUFFER
    INSERT 'cc inserted before the buffer increase'

  PROCESS ON DOMI-TEST-PROG, PLANC-MC ENDPROC
ENDCONFIG
```

The result is that the string 'cc ....' is inserted into the mode file just in front of the statement which defines the system buffer pool size. The buffer statement itself (BUFFER + <buffer increment value>) need not to be used. Without the SKIP the string would be inserted in front of the commands compiled from the PLACE statement.

Another example

```
CONFIGURATION FOR MPMStdDomino

   SKIP TO DOMAIN
      INSERT 'abort-batch-on-error off'

  ....
   .....
ENDCONFIG
```

Without SKIP the string would be inserted before the linkage-loader is called.

The following table shows where the strings are inserted when using one of the reserved words in the SKIP TO statement:

| RESERVED WORD | WHERE STRINGS ARE INSERTED |
|---|---|
| LINKER | before the linkage loader is called (remember "@" inside the strings!) |
| DOMAIN | before the domain is opened, can for example be used to release and delete the domain first |
| PLACE | before the data load address for DOMINOS is set |
| BUFFER | before the program load address for DOMINOS is set |
| SYSTEM | before the DOMINOS file(s) are loaded |
| USERADDR | before the segment is closed after having loaded DOMINOS |
| EXTENSION | after the segment is opened again and the user/UDSE data load address is established |
| PROCESS | before process files are loaded |
| ENDCONFIG | before the segment is closed and END-DOMAIN is executed |

**The compiler**

The configuration compiler is a subsystem for the ND-100 with only a few commands. Besides the standard commands HELP, EXIT and so on, the following are available:

- CONFIGURE command with parameter "source-file" (default : terminal), "mode output file" (default : terminal) and "list file" (default : no listing). Using the default values and calling the compiler inside a mode file seems the best solution. If a separate source file is used the file type :DCNF is assumed.

- Two other commands LIST-KEY-WORDS and LIST-CONTROLLER-TYPES display the reserved keywords and the possible controller types respectively.

## 5.2 DOMINOS Services

**Naming convention**

Symbolic names in ND-products shall follow a convention to reduce the risk of conflicts with user-defined names. The prefix **PI** is reserved for DOMINOS. This has led to incompatibility with PIOCOS as for instance **RealTime** has become **PIRealTime** and **U1OK PIOK**.

**Call interface**

DOMINOS has an exported routine for each service, including the user-defined services. The user has to import these routines by including the file DOMI-APPL-IE-C:IMPT into his source files, and optionally DOMI-UDSE-IE-C:IMPT for UDS. Type declarations are in the file DOMI-DEFINES:DEFS. The imported routines generally appear as follows:

```
Import (Routine <option> PIPservice, Integer2 : PIRservice)
```

**option** is a PLANC routine modifier (e.g. SPECIAL or NATIVE). The user need not worry about it as long as the routine is not called with assembler code. The option is subject to change from one DOMINOS version to another.

**service** is the name of the service (e.g. SetEv).

The **invalue** is a pointer to the parameter record unique for each service. It has the same name as the routine, except for the prefix PIP (PI Pointer), while PIR means PI Routine. There is also a corresponding record type called PIT (PI Type).

The **outvalue** is status from the service. There are predeclared constants for the different errors that can occur in the file DOMI-DEFINES.

Trapping of errors from the DOMINOS services using PLANC **On Routineerror** is not possible. If nothing else is mentioned, all out parameters in the record are undefined in case an error code is returned.

The appropriate call can be done as follows:

```
Addr <ParameterRecord> <RoutineName> =: ReturnStatus
```

NOTE! Most CPU registers (D0:D7, A0:A4) may be overwritten after the call. This is especially important in user userwritten interrupt handlers where all registers need to be saved. The PIOC compatible TRAP #2 sequence, where the registers are saved, can be used in interrupt handlers, or even better, the interrupt handler itself can save all registers on entry and restore them on exit.

The old TRAP #2 sequence of calling services in DOMINOS will remain available in the foreseeable future. It is however slower than the new way of invoking services. Only the registers D0 and SR are changed when returning from DOMINOS.

Completely new functions compared to PIOCOS have generally an additional parameter called PISubFunc (type Integer2). If nothing else is mentioned, this must be initialized to 1 (to eliminate the risk of "automatic initialization" by the compiler or loader). This parameter allows future extension of the function. For most of the functions already known from PIOCOS, some of the bits of the parameter **PIProcess** are reserved for this purpose (besides **Create**, **WhoAmI** and **ProsNo**, as PIProcess contains an out-value in these functions).

| Constant | Octal value |
|---|---|
| PITermination | 6000B |
| PIILCAL | 6001B |
| PIRANGE | 6002B |
| PICONTX | 6003B |
| PISupModeCall | 6004B |
| PIintErr | 6005B |
| PIDomFatal | 6006B |
| PIUserFatal | 6007B |
| PINOEXIST | 6011B |
| PIEXIST | 6012B |
| PIILPRI | 6013B |
| PIILSTATE | 6014B |
| PINOPROS | 6015B |
| PINOFREE | 6016B |
| PIEVNOEX | 6021B |
| PIILVEC | 6022B |
| PINOBUF | 6041B |
| PIINCONSIST | 6042B |
| PIILADDR | 6043B |
| PINoRout | 6051B |

*Table 4. DOMINOS error codes*

**Error codes**

The DOMINO Operating System errors are found in Appendix C on page 264-265.

The DOMINO Services (HW-LIB/OPCOM) error codes, are found in Appendix C on page 266.

The DOMINO Services (BOPCOM) error codes, are found in Appendix C on page 267.

## 5.3 Process Management

Processes are application programs which can run virtually in parallel under DOMINOS process management. Each process is in one of the following states:

**Dormant** Dormant means that the process exists but is completely passive.

**Blocked** Blocked means that the process is in wait state. For the time being, only "waiting for event" is a possible reason. User-defined services may create more reasons.

**Ready** The process is in the ready queue because it is ready to execute. The process is not executing because there are other process(es) with higher or equal priority in the queue.

**Running** That process in the set of **ready** processes which has the highest priority, and thus is the current executing one.

All processes which are ready for execution are linked to the **ready** queue in the order of their priority, highest priority at the head of the queue. The **scheduler** always selects process at the head of the queue, sets its state to running and executes it.

Each process has a time limit. This is the maximum CPU time it can use before being moved backwards in the ready queue. The value of the limit is determined during creation of the process and can be changed with the **Modify** service. Default (maximum) is 244 days 15 hours. The time limit is restored each time a process becomes running.

If a process stays in the **running** state when the time limit expires, it is moved behind the last process in the ready queue with the same priority. This is partial round-robin scheduling.

| |
|---|
| NOTE! The basic time unit in DOMINOS is 5 msec. |

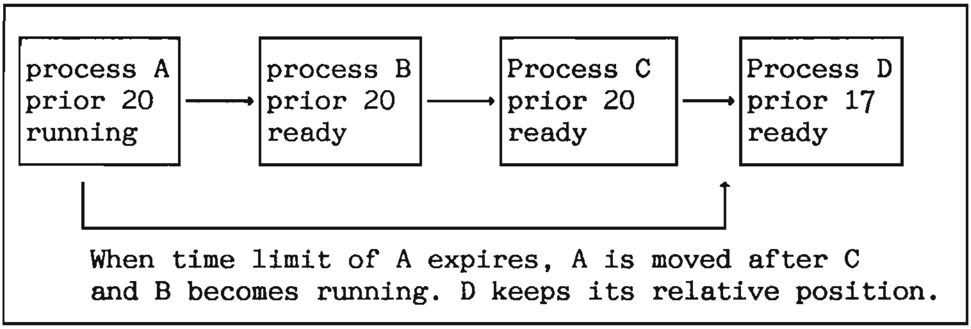



*Figure 18. Round-robin scheduling*

## 5.3.1 Create service

**Explanation** Create a process. This means in particular:

- Memory space for the process description is allocated from the system buffer pool.
- The process description is initialized according to default values and parameters given by the service.
- A new entry is allocated in the process table and given to the process description.
- The time limit is set to maximum (244 days, 15 hours).
- If the "create and go" option is selected, the process is set into the **ready** state, otherwise it becomes **dormant**.

| IN | PITCreate record | OUT |
|---|---|---|
| | PIProcess | → Integer2 |
| Bytes → | PIName (0:3) | |
| Integer2 → | PITypePrio | |
| Program Pointer → | PIStartAddr | |

**PIProcess** The process number is returned. Most process-related functions need the value as input. The number of processes that may be created is mainly limited by buffer space in DOMINOS.

**PIName** User-chosen process name of four characters. If a shorter name is used, fill up with spaces. It must not be in use by another process.

**PITypePrio** PITypePrio contains in fact several parameters:

- The **PIFullAcc** bit defines whether the process has write access to the device-dependent IO-space of memory or not. If set, the process is allowed to do so, and if not, only user data is accessible. However, the memory protection is global for all processes. If one process has access to the IO-space, the IO-space cannot be protected. **Thus this option should not be used unless it is absolutely necessary!**

- If the "create and go" bit is set to one (**PIABegin**), the process is started, otherwise the service **Begin** has to be called explicitly.

- Bits 0:7 contain the process priority within the range 1:255. A process with priority 0 is illegal, while 255 is reserved for future extension.

- All other bits must be zero in order not to conflict with future extensions of the create service.

**PIStartAddr** Points to the first instruction to be executed by this process after it has been started.

Example of starting a process

```
$Include DOMI-DEFINES:DEFS

Module Common
   Export ErrCheck, WaitSeconds
   % as in previous example
Endmodule

Module PRO2
   Export PRO2
   % as in previous example
Endmodule

Module Auto_Start
   Export Auto_Start
   $Include DOMI-APPL-IE:IMPT
   Import (Routine Integer, Void : ErrCheck)
   Import (Program : PRO2 )
   Integer Array : S (0:1023)
   Constant Prior = 13
PITCreate : CreRec := (0,'PRO2',PIABegin+Prior,Addr(PRO2))
   Program : Auto_Start
      IniStack S
      Output(1,'A','$Auto_Start is creating PRO2')
      Addr (CreRec) PIRCreate ErrCheck
   Endroutine
Endmodule
```

## 5.3.2 Modify service

**Explanation**

The **Modify** service can change the parameters of a process. The parameter record is a variant record of **PITCreate**. The process number is **IN** parameter and specifies the process to be changed.

All parameters supplied in the **Create** service can be modified. The process name, memory access and priority affect the process immediately, while start address will be used when the next **Begin** service is called for the process.

If the new process priority is equal to the old one, and the process being modified is actually the running one, the process state is affected. The process is moved in the ready queue behind all processes with same priority.

The extra parameter **PITimeLimit** is used to give an individual time limit to the process for round-robin scheduling. That is the maximum number of CPU time units to have exclusive access to the CPU. The parameter is interpreted as a 32-bit unsigned integer. Zero means in fact the maximum time limit, which is used when the process is created.

PITModify record of PITCreate

| | IN | | OUT |
|---|---|---|---|
| Integer2 | → | PIProcess | |
| Bytes | → | PIName (0:3) | |
| Integer2 | → | PITypePrio | |
| Program Pointer | → | PIStartAddr | |
| Integer4 | → | PITimeLimit | |

Example of a process which changes its priority

```
Module PRO2
 Export ModRec  % used by Auto_Start when creating process
 $Include DOMI-APPL-IE:IMPT
 Import (Routine Integer, Void : ErrCheck)
 Integer Array : S (0:1023)
 Program : PRO2?  % predeclaration
 PITModify : ModRec := (0, 'PRO2', 0, Addr(PRO2), 0)
 Program : PRO2
    IniStack S
    Output(1,'A','$PRO2 running with unknown priority')
    Output(1,'A','$I will now set it to 10')
    10 =: ModRec.PITypePrio
    Addr (ModRec) PIRModify ErrCheck
 Endroutine
Endmodule

Module Auto_Start
   Export Auto_Start
   Import (PITModify : ModRec)
   % code as for previous example, except using ModRec
Endmodule
```

## 5.3.3 Begin service

**Explanation** Set a process to the **ready** state. This means in particular that the scheduler can take care of it, and start execution of it (set it into the **running** state) as soon as there is no other process ready with higher priority. The start address is the one given when the process was created if it has not been modified in the meantime. See also the services **END** and **ABORT**.

PITBegin record

IN OUT

Integer2 → PIProcess

Example of creating and starting a process by two services

```
Module PRO2
   Export PRO2
   % code as before
Endmodule

Module Auto_Start
   Export Auto_Start
   $Include DOMI-APPL-IE:IMPT
   Import (Routine Integer, Void : ErrCheck)
   Import (Program : PRO2 )
   Integer Array : S (0:1023)
   Constant Prior = 13
   PITCreate : CreRec := (0, 'PRO2', Prior, Addr(PRO2) )
   PITBegin  : BegRec := (0)
   Program : Auto_Start
      IniStack S
      Output(1,'A','$Creating PRO2 without starting it')
      Addr (CreRec) PIRCreate ErrCheck
      Output(1,'A','$Starting PRO2')
      CreRec.PIProcess =: BegRec.PIProcess
      Addr (BegRec) PIRBegin
  Endroutine
Endmodule
```

### 5.3.4 End service

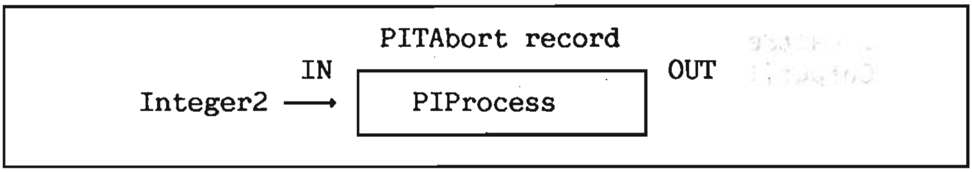
The process executing this call is taken from the **running** to the **dormant** state. Only a new **BEGIN** service (issued by a different process) can start it again. All entries in the timer queue concerning the dormant process are removed. The pointer being the in-value to this service is dummy, so **NIL** should be used.

Example of a process which stops itself

```
Module PRO2
   $Include DOMI-APPL-IE:IMPT
   Import (Routine Integer, Void : ErrCheck)
   Export PRO2
   Integer Array : S (0:1023)
   Program : PRO2
      IniStack S
      Output(1,'A','$PRO2 running')
      Output(1,'A','$I will now make myself dormant')
      Nil PIREnd ErrCheck  % Usually never reached !!
   Endroutine
Endmodule

Module Auto_Start
   Export Auto_Start
   % code as before
Endmodule
```

## 5.3.5 Abort service

This service has the same effect as **END**, but the process to be made dormant is specified in the parameter record.

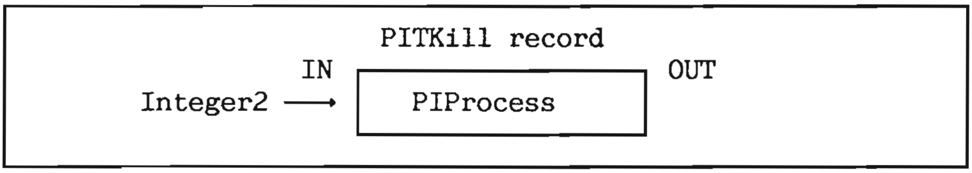


**Example of starting and stopping a process several times**

```
Module PRO2
   ...
   Integer Array : S (0:1023)
   Program : PRO2
      IniStack S
      Do % infinite loop
         1 WaitSeconds
         Output(1,'A','$PRO2 running')
      Enddo
   Endroutine
Endmodule

Module Auto_Start
  ...
  Integer Array : S (0:1023)
  Constant Prior = 6
  PITCreate : CreRec := (0, 'PRO2', Prior, Addr(PRO2) )
  PITAbort  : AboRec := (0)
  PITBegin  : BegRec := (0)
  Program : Auto_Start
   IniStack S
   Output(1,'A','$Creating process PRO2')
   Addr (CreRec) PIRCreate ErrCheck
   CreRec.PIProcess =: AboRec.PIProcess =: BegRec.PIProcess
     Do
         Output(1,'A','$Starting process PRO2')
         Addr (BegRec) PIRBegin ErrCheck
         5 WaitSeconds
         Output(1,'A','$Stopping process PRO2')
         Addr (AboRec) PIRAbort ErrCheck
         3 WaitSeconds
      Enddo
   Endroutine
Endmodule
```

## 5.3.6 Kill service

The process given in the parameter record is made dormant. The process description is returned to the buffer pool and its entry in the process table is cleared. The process no longer exists (i.e. its process number becomes undefined).



**Example of creating and deleting a process several times**

```
Module PRO2
   ...
   Integer Array : S (0:1023)
   Program : PRO2
      IniStack S
      Do While True
         % infinite loop
         1 WaitSeconds
         Output(1,'A','$PRO2 running')
      Enddo
   Endroutine
Endmodule

Module Auto_Start
 ...
 Integer Array : S (0:1023)
 Constant Prior = 2
 PITCreate : CreRec :=(0,'PRO2',PIABegin+Prior,Addr(PRO2))
 PITKill   :  KilRec  := (0)
 Program : Auto_Start
    IniStack S
    Do
       Output(1,'A','$Creating process PRO2')
       Addr (CreRec) PIRCreate ErrCheck
       7 WaitSeconds
       Output(1,'A','$Deleting process PRO2')
       CreRec.PIProcess =: KilRec.PIProcess
       Addr (KilRec) PIRKill ErrCheck
       4 WaitSeconds
    Enddo
 Endroutine
Endmodule
```

## 5.3.7 WhoAmI service

Obtain the process number of the calling process. The process number is needed as parameter in several services.

| | PITWhoAmI record | |
|---|---|---|
| IN | | OUT |
| | PIProcess | → Integer2 |

**Example of a process calculating its process number**

```
Module PRO2
   ...
   Integer Array : S (0:1023)
   PITWhoAmI : WhoRec := (0)
   Program : PRO2
      IniStack S
      Output(1,'A','$PRO2 running with process number ')
      Addr (WhoRec) PIRWhoAmI ErrCheck
      Output(1,'I',WhoRec.PIProcess)
   Endroutine
Endmodule
```

## 5.3.8 ProsNo service

Obtain number of a process with a given name.

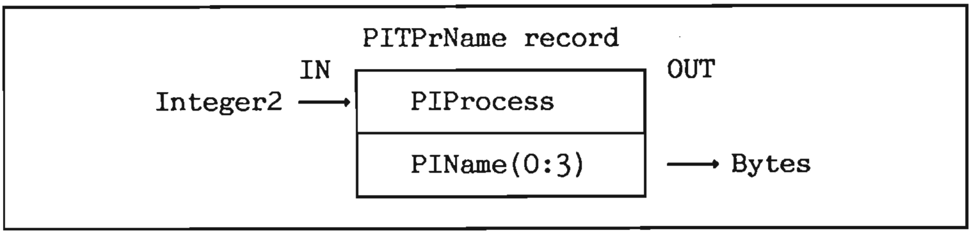
```
                  PITProsNo record
          IN    +------------------+  OUT
                |    PIProcess     |  ---> Integer2
                +------------------+
Bytes   ------> |   PIName(0:3)    |
                +------------------+
```

**Another example of how to get the process number**

```
Module PRO2
   ...
   Integer Array : S (0:1023)
   PITProsNo : ProRec := (0,'PRO2')
   Program : PRO2
      IniStack S
      % I know my name, but not my process number
      Addr (ProRec) PIRProsNo ErrCheck
      Output(1,'A','$PRO2 running with process number ')
      Output(1,'I',ProRec.PIProcess)
   Endroutine
Endmodule
```

## 5.3.9 PrName service

Obtain name of a process with a given process number.




Example of getting the process name

```
Module PRO2
  ...
  Integer Array : S (0:1023)
  PITPrName : PrNRec := (0,'    ')
  PITWhoAmI : WhoRec := (0)
  Program : PRO2
     IniStack S
     % I neither know my process number nor name
     Addr (WhoRec) PIRWhoAmI ErrCheck
     % I still do not know my name
     WhoRec.PIProcess =: PrNRec.PIProcess
     Addr (PrNRec) PIRPrName ErrCheck
     Output(1,'A','$')
     Output(1,'A',PrNRec.PIName)
     Output(1,'A',' running with process number ')
     Output(1,'I',PrNRec.PIProcess)
  Endroutine
Endmodule
```

## 5.4 The Event System

**Event system**

The event system is used for synchronization purposes between processes (two way synchronization), and between interrupt handlers and processes (one way synchronization). It is a very general concept which may used for solving a broad range of problems regarding the signalling part of interprocess communication.

| Driver to process (one way) | Process to process(both ways) |
|---|---|
| Interrupt<br>$\downarrow$<br>Driver $\xrightarrow{\text{event}}$ Process | Server process $\xrightarrow{\text{Ack event}}$ Client process<br>Server process $\xleftarrow[\text{Req event}]{}$ Client process |

**Event buffer**

Each process has an event buffer containing the current events set for it. The buffer is an integer variable where each bit corresponds to a discrete event.

**Event agreement**

The communicating processes must agree in advance upon which events (bits) to use and upon their semantic values. Events are normally used in combination with additional information exchanged between the processes (e.g. a message residing in a mailbox). The event just says that 'something has occurred', but not what or which process caused it.

A process can set events for any other process (including itself) as long as it knows it's process number. Events can only be sent to processes in the same DOMINO controller. NUCLEUS must be used if communication with remote processes is needed. NUCLEUS uses events 30 and 31 (decimal bit number).

## 5.4.1 SetEv service

Set event(s) for a process. The events given by **PIEvents** are added (ORed) to the event buffer of the receiving process. There is no event queuing in case some of the events already are set for it. An event may only be sent to one process at a time (no broadcast possibility).

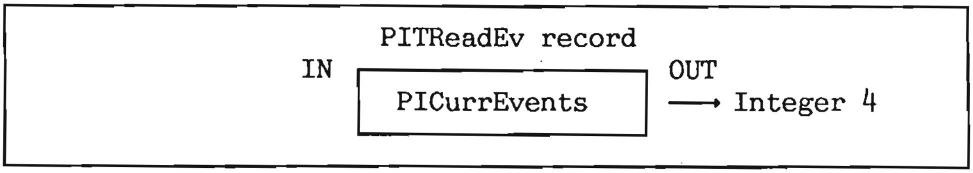
This can lead to events being lost (overwritten) in the receiver's event buffer if the receiving process is slow compared to the senders. Careful design of an event protocol between the communicating processes removes the problem.

PITSetEv record

| IN | | OUT |
|---|---|---|
| Integer2 → | PIProcess | |
| Integer4 → | PIEvent | |

## 5.4.2 ReadEv service

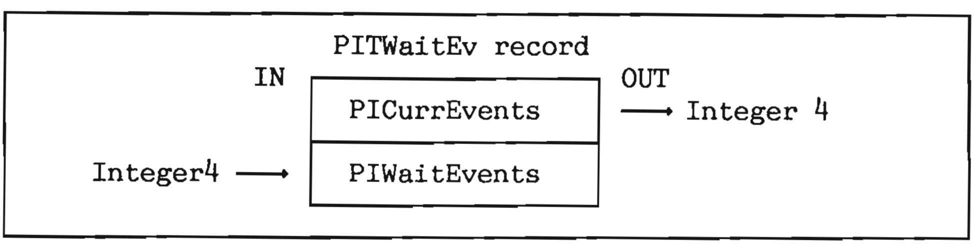
The events currently set for the process are returned in the variable **PICurrEvents**. The process will return immediately (never enter blocked state) even if there are no events. The returned events are cleared in the process' event buffer.




## 5.4.3 WaitEv service

The process will wait for events given by the variable **PIWaitEvents**. The process continues when any of these events are set (one or more). **All** events set are then returned in the variable **PICurrEvents**, and the event buffer of the process is cleared.

If the events to wait for are equal to zero, the process will never return. If there are already events in the buffer which match PIWaitEvents when **WaitEv** is called, the process continues immediately.



Example of signalling ('kicking') a process

```
MODULE MASTER %This example is only relevant when loss of
EXPORT MASTER %events is acceptable, or there is a certain
              %maximum delay before the receiver reacts.
PITProsNo : ProRec := (0,'SLAV')

PROGRAM : MAST    % Master process (kicking slave)
PITSetEv : SetEvRec
INTEGER : KickCount
..
% wait for creation of SLAVE process
ProRec.PIProcess =: SetEvRec.PIProcess  % ProRec of slave
 1 =: SetEvRec.PIEvent; 0 =: KickCount  % Bit/Event 0 set

DO
  ++ KickCount
  Output(1,'A','$MAST : Kicking SLAV for ')
  Output(1,'I',KickCount)
  Output(1,'A','th time')
  Addr(SetEvRec) PIRSetEv              % kick partner
  1 WaitSeconds                        % ensure slave gets
ENDDO                                  % time to react
ENDROUTINE
ENDMODULE

MODULE SLAVE
EXPORT SLAVE

PROGRAM : EVE2
PITWaitEv : WaitEvRec
INTEGER : KickedCount
..
0 =: KickedCount
% return when any event occurs
-1 =: WaitEvRec.PIWaitEvents      % return when any event
DO
  Addr(WaitEvRec) PIRWaitEv
  ++ KickedCount
  Output(1,'A','$EVE2 : I have been kicked for ')
  Output(1,'I',KickedCount); Output(1,'A','th time')
ENDDO
ENDROUTINE
ENDMODULE
```

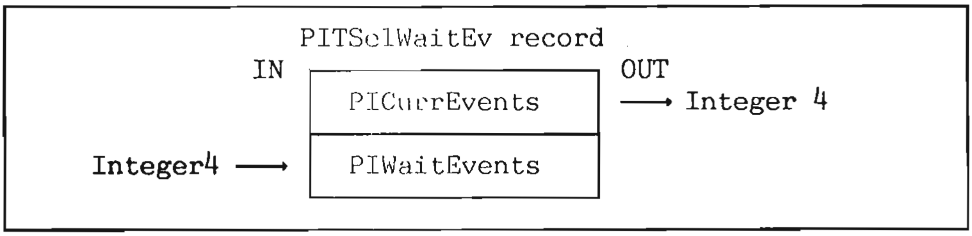
Example of signalling events with positive feedback

```
MODULE MASTER  % The kicking process waits for acknowledge
EXPORT MASTER  % from the slave in this example. If the
               % slave dies, the master is stuck!

TYPE EventKind = ENUMERATION (None,Kick,Ack)
PITProsNo : ProRec := (0,'EVE2')
PITSetEv  : SetEvRec

PROGRAM : MASTER
  PITWaitEv : WaitEvRec
  PITSetEv  : SetEvRec
  INTEGER   : KickCount
  ..
  ProRec.PIProcess =: SetEvRec.PIProcess
  Kick CONVERT INTEGER =: SetEvRec.PIEvent % prepare kick
  0 =: KickCount; -1 =: WaitEvRec.PIWaitEvents

 DO
    ++ KickCount
    Output(1,'A','$MAST : Kicking SLAV for ')
    Output(1,'I',KickCount); Output(1,'A','th time')
    Addr(SetEvRec) PIRSetEv ErrCheck    % kick slave
    Addr(WaitEvRec) PIRWaitEv ErrCheck  % wait acknowledge
    CASE ( WaitEvRec.PICurrEvents CONVERT EventKInd )
    INCASE Ack
       Output(1,'A','$MAST : Ack')
    ELSE % protocol violation
    ENDCASE
ENDDO
ENDROUTINE
ENDMODULE
```

```
MODULE SLAVE
EXPORT SLAVE
TYPE EventKind = ENUMERATION (None,Kick,Ack)

PROGRAM : SLAVE
  PITSetEv  : SetEvRec
  PITWaitEv : WaitEvRec
  INTEGER : KickedCount
  ..
   ProRec.PIProcess =: SetEvRec.PIProcess
   0 =: KickedCount; -1 =: WaitEvRec.PIWaitEvents

   DO % forever
    Addr(WaitEvRec) PIRWaitEv ErrCheck  % wait acknowledge
    CASE ( WaitEvRec.PICurrEvents CONVERT EventKInd )
    INCASE Kick
       ++ KickedCount
       Output(1,'A','$SLAV : I have been kicked for ')
       Output(1,'I',KickedCount); Output(1,'A','th time')
       Ack CONVERT INTEGER =: SetEvRec.PIEvent
       Addr(SetEvRec) PIRSetEv  ErrCheck
    INCASE Ack
       Output(1,'A','$SLAV : Ack')
    ELSE %  protocol violation
    ENDCASE
 ENDDO
 ENDROUTINE
 ENDMODULE
```

## 5.4.4 SelWaitEv service

This service does basically the same as WaitEvents. The difference being that only those events the process is waiting for are cleared in the event buffer. **PICurrEvents** contains however <u>all</u> events at return (selective wait).




## 5.4.5 UniWaitEv service

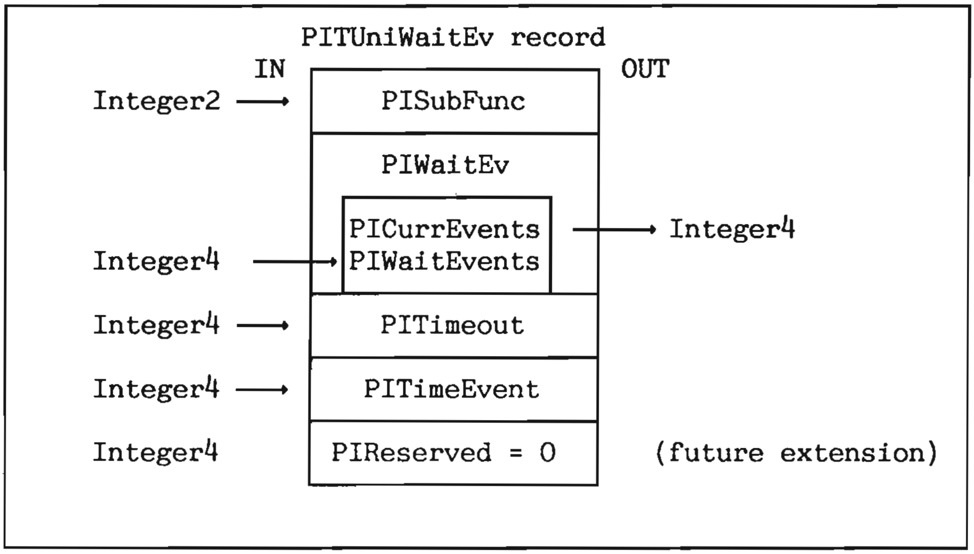
The UniWaitEv (universal) service provides all the functions that are possible with **SelWaitEv** and **WaitEv**, and some additional.

The parameter **PISubFunc** parameter may contain the sum of any of the constants PIEvSel and PIEvComp.

**PIEvSel** Only those events the process is waiting for (WaitEvents parameter of PITWaitEv) are returned in the variable **PICurrEvents** and cleared in the event buffer.

**PIEvComp** The process is continued only when **all** the events given by WaitEvents parameter have occurred.

**PITimeOut** If the PITimeOut parameter is different from zero, this indicates the time to wait (basic time units) before giving the process timeout. The event(s) given by PITimeEvent are returned in this case. A bit should normally be reserved for signalling timeout. Otherwise it will not be possible to distinguish timeout from other events as no more distinguish timeout from other events as no more context accompanies it. If any of the WaitEvents occur before timeout, the process returns and the timeout is cancelled.




**Example of halting a process for a given time**

```
PITUniWaitEv : WaitUniRec := (PIEvsel,(0,1),200,1,0)

ROUTINE INTEGER, VOID : WaitSeconds
 1 =: WaitUniRec.PIWaitEv.PIWaitEvents  % wait for timeout
 @ * 200 =: WaitUniRec.PITimeOut % set timeout event
 Addr (WaitUniRec) PIRUniWaitEv ErrCheck % wait
ENDROUTINE
```

## 5.5 Time Scheduled Events

**Timer queue** DOMINOS uses a timer queue in order to provide time-scheduled events. Each entry in the queue gives information about the process requesting the service, and when it wants it.

**PIOCOS** The clock process in PIOCOS is replaced by a more sophisticated clock driver in DOMINOS. The length of the timer queue is only limited by the size of the system buffer pool.

### 5.5.1 InterEv service

This service provides time related events at regular intervals.

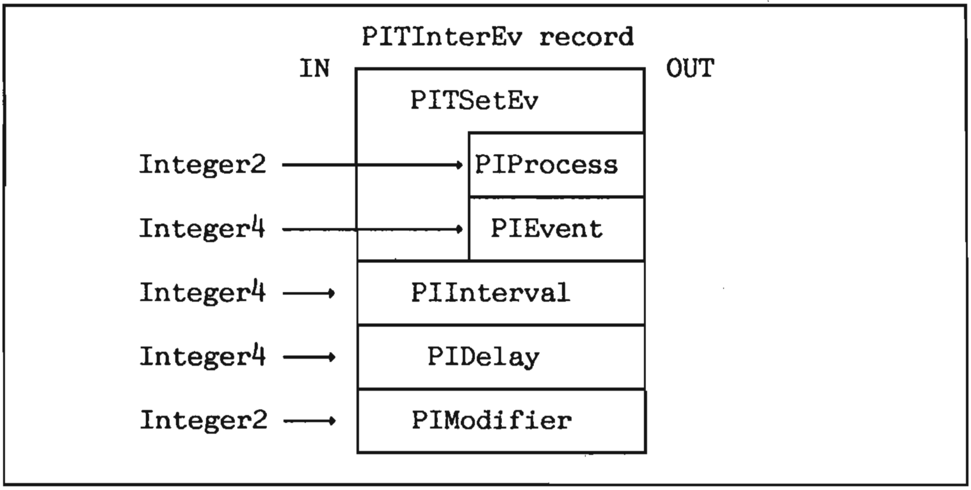
**PIDelay** After a number of time units, given by **PIDelay**, the events in the PIEvent parameter are set. The value 0 (zero) is invalid.

**PIInterval** If PIInterval is different from zero, it gives the delay interval before the events are set again. The events are set repeatedly until the service is cancelled by the **InterDel** service.

**PIModifier** PIModifier may contain the sum of any of the constants PIC1Second and PIC1Relative.

**PIC1Sec** The parameters **PIInterval** and **PIDelay** are by default taken to be given in basic time units. This constant changes the time unit to seconds.

**PIC1Rel** By default, the delay before setting the events are scheduled absolute to the DOMINOS system clock. This constant causes the event to be scheduled relative to the time when this service is called. The DOMINOS clock is exported in the parameter PIRealTime into the application program.

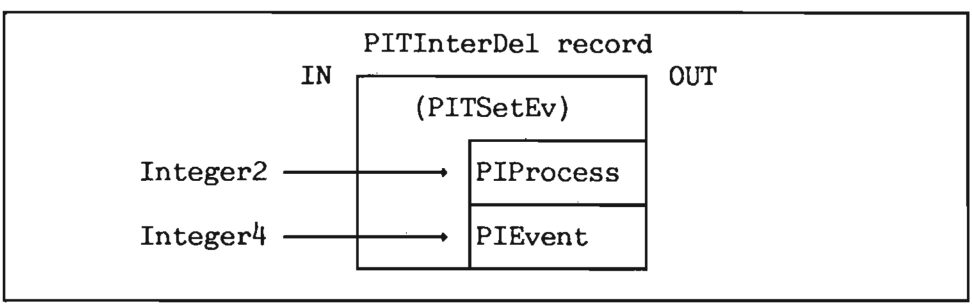



## 5.5.2 InterDel service

Each requested **InterEv** service can be cancelled with this call. The parameter **PIEvent** cancels services according to the following rule:

- If the PIEvent is equal to zero, all time services belonging to PIProcesses are cancelled.

- If PIEvent >< 0 then all time services matching both the PIEvent and the PIProcess parameter are cancelled.



## 5.6 Buffer Management

**Buffer pool** The buffer management supplies utility functions for administrating shared pools between the processes. There is one pool for the user processes and another only accessible from supervisor mode (e.g. DOMINOS and device drivers). The buffer management is of general purpose. The data kept in a buffer is not interpreted by DOMINOS. A common pool makes it possible to save memory space, as it is no longer necessary to allocate one buffer heap per process. A buffer in the pool is at any time either free to be used, or allocated to a process. The use of the pools is allocated at load time.

**PIOCOS** The buffer management was also implemented in PIOCOS. However, the services have been revised to include a pool of user buffers and are now available for the user too.

**Memory layout** The system buffer pool is defined by the memory gap between data and code part of DOMINOS. The user buffer pool is defined by the global label **PIUserBuffer** and the start of the memory area for the system (supervisor) stack. This stack is located in the last part of the local memory.

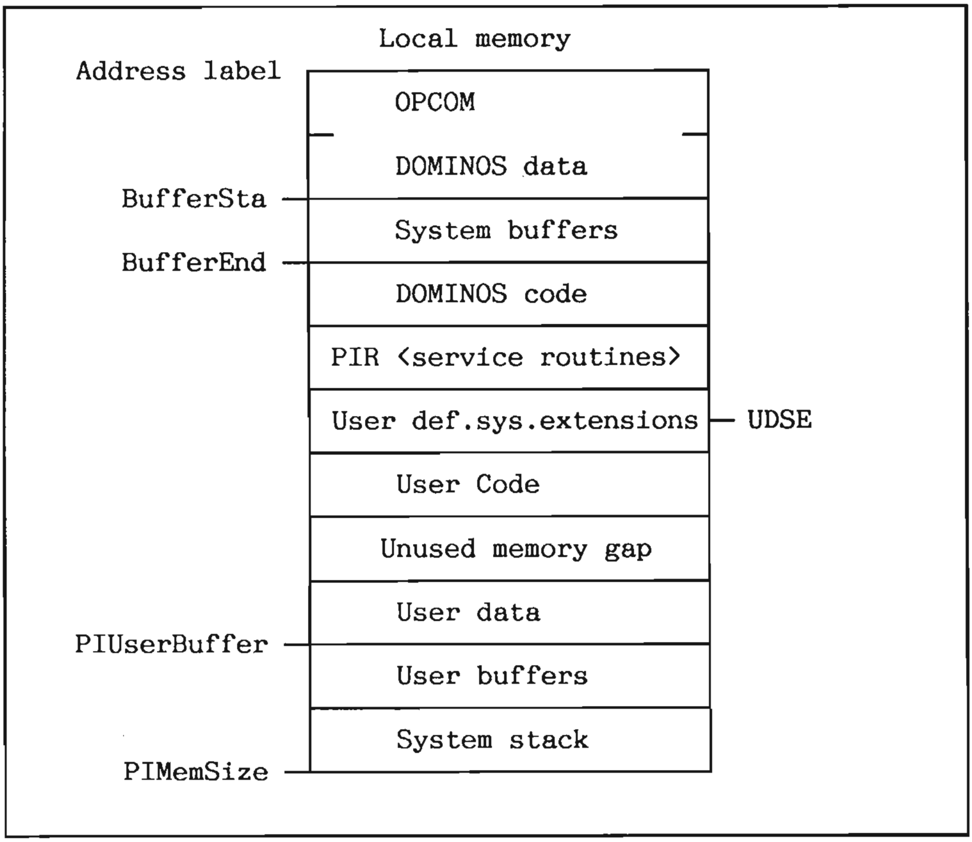



*Figure 19. DOMINOS relative memory layout*

**Consistency check**

The data structure for the buffer management is a double-linked list of elements. Each element keeps information about one buffer. The elements are ordered according to increasing memory address. If a buffer is given to a process, a flag is set in the corresponding list element. When DOMINOS starts, there is one list element spanning the whole pool. As buffers are given to processes, the number of list elements increases (fragmentation). The buffer list is checked for consistency. Inconsistency during DOMINOS start, leads to a fatal system error (DOMINOS is aborted).

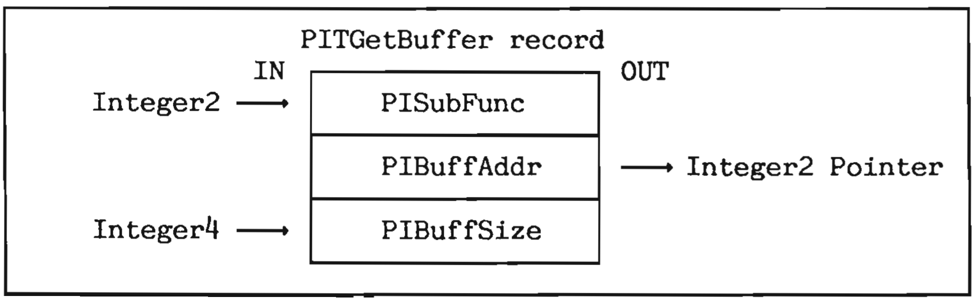
## 5.6.1 GetBuffer service

Allocate a buffer to a process. If there is not enough free buffer space, an error message is returned.

**PISubFunc** This modifier indicates in which pool the buffer resides. Use **PIUsrBuff** for user pool and **PISysBuff** for system pool.

**PIBuffAddr** The address (32-bit pointer) to the start of the buffer. Returned by DOMINOS.

**PIBuffSize** The size of the desired buffer in bytes.

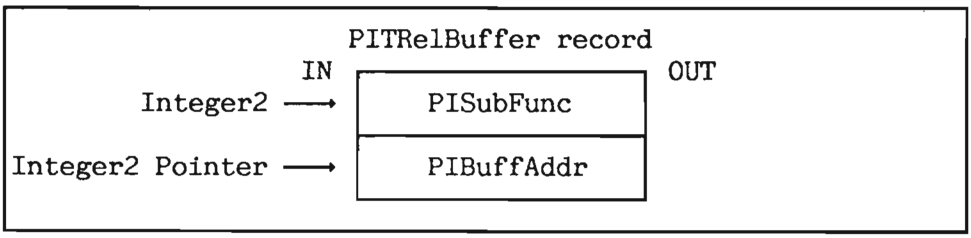


## 5.6.2 RelBuffer service

Release a buffer allocated to a process. The service checks for consistency.

**PISubFunc** This modifier indicates in which pool the buffer resides. Use **PIUsrBuff** for user pool and **PISysBuff** for system pool.

**PIBuffAddr** The address (32-bit pointer) to the start of the buffer.




## 5.7 Exported system data

Some data items are exported from DOMINOS and can be imported to application modules. Note that all these data items reside in a memory area which is write-protected. All the data items can be replaced in a later version of DOMINOS by routines with equal names and a corresponding out value.

**PIRealTime** An INTEGER4 variable which is incremented by the clock driver on each timer interrupt (every 5 msecs). It is initialized to 0 on DOMINO startup.

**PIHostNumb** The number of the host CPU as an INTEGER2 variable. For DOMINO controllers the contents of the variable is not currently defined.

**PIControll** The number of the controller in the current host system as an INTEGER2 variable. The contents on DOMINO is the OCTOBUS Station number of the controller.

**PICPUType** INTEGER4 variable which contains a number describing the processor used on this controller. The following values are defined for the MOTOROLA processors (decimal) : 68000, 68010 and 68020.

## 5.7.1 Fatal service

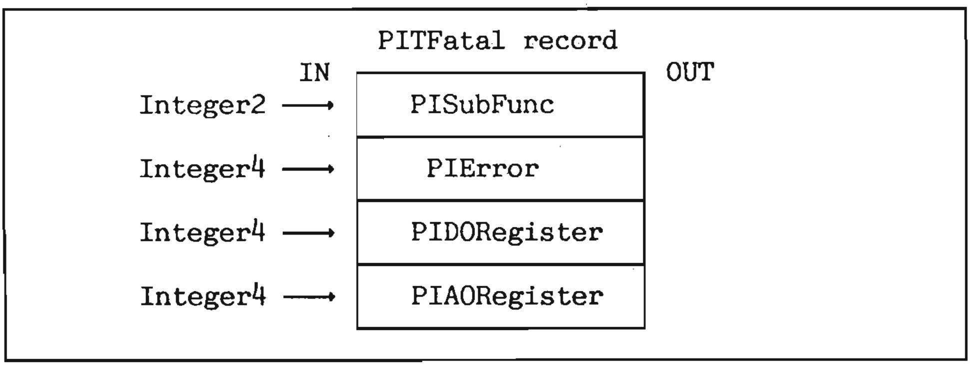
If a fatal error occurs in a user program, this error can be reported to the host by using this call. Note that DOMINOS is aborted. This means **all activities are stopped and the controller must be loaded and started again!**

This ultimate service can be called from interrupt handlers too.

**PISubFunc** For future extension. Must be set to 1.

**PIError** The user-defined error code to return. Extended to be an INTEGER 4, but only the lower 16 bits are used.

**PIA0Register & PID0Register** The parameters PIA0REGISTER and PID0REGISTER are handled such that the value of PIA0Register is put into the A0-register and the PID0Register into the D0-register. Thus, these values may be inspected by the LOOK-AT-REGISTER command in the DOMINO Monitor. The current values of A0 and D0 should therefore be saved in the record before preparing the call.




## 5.8 DOMINOS for advanced programmers

DOMINOS also offers special program environments for user defined:

- interrupt handlers
- trap handlers
- services (UDS)
- process management extensions (PME)

**UDSE**

All these entities are hereafter referred to as User-Defined System Extensions (UDSE).

## 5.8.1 The MC68K in supervisor mode

**Execution modes**

When making UDSE code, the user must be aware that the instructions will be executed in the so-called supervisor mode of the MC68K. DOMINOS executes in this mode. This is indicated by a bit in the status register (SR) of the processor and has the following implications compared with the user mode used in the process environment:

- a different machine stack is used
- there are no restrictions on the processor instruction set
- different access rights apply throughout the first 16-MByte address range

**Stack pointers**

The A7-register is used as stack pointer in the MC68K processors. The user and the supervisor mode have separate A7-registers. Depending on the execution mode, one of them is selected when A7 is accessed in an instruction.

However, it is possible in supervisor mode to read the A7-register of the user mode by using the 'MOVE from/to USP' instruction. This is necessary to let DOMINOS switch from one process to another.

**Privileged instructions**

The privileged instructions which can be executed in supervisor mode but which lead to a trap if tried in user mode are:

- All instructions which may change SR :

  MOVE to/from SR

  ANDI to SR, also EORI, ORI

  RTE

  STOP

- Coprocessor instructions cpSAVE and cpRESTORE
- RESET
- MOVEC (to access special CPU registers), MOVES (to access 'unnatural' address spaces) and MOVE to/from USP (to access the user stack pointer)
- The RTE instruction is always the last instruction to be executed in an interrupt/trap handler to resume the suspended activity.

## 5.8.2 Disable and enable interrupts

The following code shows how to turn interrupts off and on. The interrupt should normally only be turned off for short intervals.

**Code to disable all maskable interrupts**

```
INTEGER2 : SaveSR

$* MOVE.W  SR, SaveSR  % keep current value of SR
$* ORI.W   #0700H,SR   % set interrupt threshold to
  .......              % maximum uninterruptible code
$* MOVE.W  SaveSR, SR  % interrupts switched on again
```

The combination of these two pieces of code can with advantage use the stack instead of the INTEGER2 variable. The well-known rules for how to use a stack must then be followed:

**Macros to enable and disable interrupts**

```
$MACRO solo
  $* MOVE.W  SR,-(A7)    % save old SR on stack
  $* ORI.W   #0700H,SR   % disable all maskable interrupts
$ENDMACRO

$MACRO tutti
  $* MOVE.W  (A7)+,SR    % restore SR
$ENDMACRO
  ...
  ...
  SOLO                 % from now on interrupts are disabled
  ...                  % critical section with no interrupt
  TUTTI                % interrupts on again
```

**Note** Some interrupts can not be disabled, e.g. Power fail.

## 5.8.3 Access rights in supervisor mode

The hardware-based access protection system in DOMINO controllers depends on the MC68K mode of operation. Different areas inside the first 16 MBytes of the physical address range must be used for different purposes. Before DOMINOS is started, the protection is setup by the OPCOM module. Later, DOMINOS changes parts of the access map.

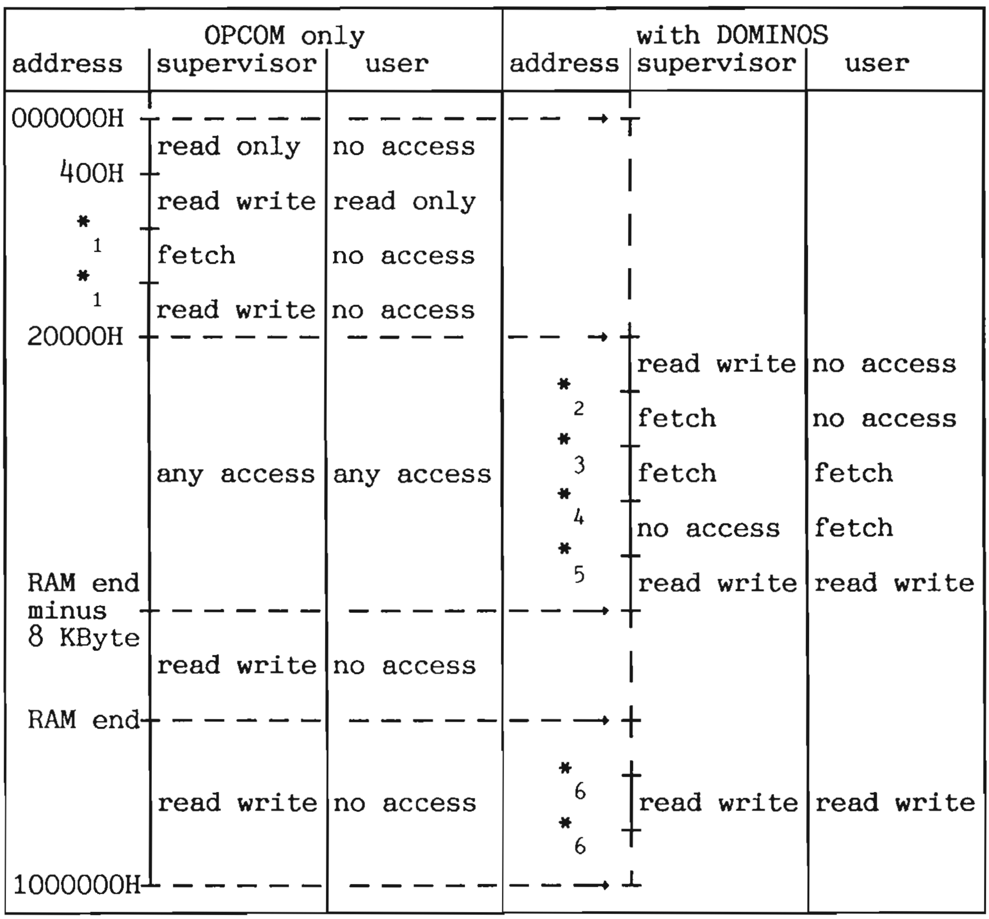



*Figure 20. DOMINO memory protection*

Notes:

$*_1$ Value depends on the current OPCOM version

$*_2$ Value depends on DOMINOS version and size of system buffer pool. See DOMINOS configurator.

$*_3$ Value depends on $*_2$ and the current DOMINOS version

$*_4$ Value depends on $*_3$ and the code part of the UDSE

$*_5$ Value depends on <load address for user/UDSE data>. See DOMINOS configurator.

$*_6$ Value depends on <sys proc extra READ/WRITE>. See DOMINOS configurator.

## 5.8.4 PLANC compiler

**Clean code & Option 2**

It is very important to use the right PLANC compiler for MC68K and to use it correctly:

- In version G one must use the compiler directive **OPTION 2** which forces the compiler to use a new calling sequence, which has no data placed in the code area (dirty code). It is also much faster. faster. From version H this option is switched on by default.

- From version H it is safer to use high level PLANC statements in SPECIAL routines: A warning is issued when the compiler generates code in a SPECIAL routine which assumes the existence of a stack (usually not present!).

- Version I should be used since the UNSIGNED modifier is used in DOMI-DEFINES-:DEFS.

**Exception handler**

Exception handler is used as a common name for interrupt handlers (asynchronous exception) and trap handlers (synchronous exception). The term 'exception' means that the CPU is forced to leave its normal execution sequence to execute some exceptional code.

**PLANC constraints**

Exception handlers are activated entirely by hardware, and they do not therefore fit into the PLANC environment. When an ordinary PLANC routine is called, the PLANC run-time system allocates a stack frame for the routine. The code inside the routine assumes that the stack is present, which is not the case for exception handlers. There are, however, three kinds of routines in PLANC which do **not** implicitly assume the presence of a stack:

- PROGRAM
- NATIVE (only available in MC68K PLANC)
- SPECIAL

**Program**

**PROGRAM** defines a main program which begins always with an **INISTACK** statement, thus making its own stack. It would therefore be a perfect solution for the problem, but there are some drawbacks:

- INISTACK must be the first statement in the routine, and the generated code destroys the register context before it can be saved.

- Only a static allocated stack is accepted in the INISTACK statement (the stack cannot reside in a heap). If interrupts on different levels use the **same handler**, the **same array** could be initialized twice, thus destroying (overwriting) the stack of the exception handler on the lower interrupt level.

- The new stack to be created is the supervisor stack. Changing this stack might have consequences for the whole system.

- A design goal for exception handlers is to make them fast and short (few instructions). The INISTACK and all the other necessary actions are quite a big overhead in many of these cases.

The conclusion is that a solution with PROGRAM is **not** recommended!

**native**

NATIVE routines are only for MC68K PLANC. They have an automatically included calling sequence based only on the MC68K **machine** stack. Registers are not destroyed when the routine is called. There are however serious compatibility problems such as that NATIVE routines must never call a normal routine (e.g. a routine in the PLANC run-time system). In addition to this, NATIVE routines are unable to use ERRETURN, and, with the exception of invalue and outvalue, have NO parameters. NATIVE-type routines are not fully supported. There is therefore no guarantee that they will not be removed from the compiler at some time in the future.

A solution with NATIVE routines can no longer be recommended since the MC-PLANC compilers now have better support for SPECIAL routines.

**Special**

The routine option SPECIAL defines a routine with no call-sequence. In practice this leads to routines which:

- have no stack frame (no parameters and local data)
- are rather fast

Earlier it was quite dangerous to use anything else than pure inline assembler in such routines since the generated code assumed that a stack frame existed. The latest versions of the MC-PLANC compiler issue a warning when it uses the stack frame in a SPECIAL routine. High-level PLANC statements in SPECIAL routines are therefore now possible which allows the implementation of increased complexity.

This type of routine can with advantage be used in cases where the complexity of the handler is small or medium.

**Mixed routines** The recommended solution for complex exception handlers is a combination of one SPECIAL routine and normal routines.

**INISTACK simulation** It is quite easy to simulate an INISTACK statement for the stack layout belonging to the "OPTION 2", using an array **dynamically** allocated on the machine stack! This is exactly what the standard INISTACK lacks!

Assembler code for saving registers and allocate stack

```
PITDriver : AnExceptHandler   %predeclared SPECIAL routine
 $* MOVEM.L D0-A5, -(A7)      %save user registers

 $* LINK    A6,#<stack demand>%save A6 and allocate stack
 $* MOVE.L  A6,4B(A7)         %save former A7
 $* PEA     14B(A7)           %generate FREE pointer
 $* MOVE.L  A7,A6             %is now PLANC stack pointer
```

The code inside the exception handler is now free to use the address and data registers. The new stack frame is now initialized with a size of the absolute value of <stack demand> given in bytes. The parameter <stack demand> must be given as a negative argument to fit with the MC68K. The size of the stack need only take care of the routines called (directly or indirectly) from here. In the current version, the value should not exceed 1 KByte. Ordinary PLANC routines (with or without parameters) can be called. Note that the current SPECIAL routine must still not have any local variables!

Having executed the routines of the exception handler, the original context must be restored:

| Assembler code for deallocating stack and restoring registers |
|---|

```
 $* MOVE.L  10B(A6),A7          % deallocate current stack
 $* MOVE.L  (A7)+,A6            % restore A6
 $* MOVEM.L (A7)+,D0-A5         % restore user registers
 $* RTE                         % resume interrupted activity
ENDROUTINE                      % (pop machine stack)
```

Note that the INISTACK simulation here is dependent on the current implementation of the stack layout. It may therefore change in the future!

In cases where the user can guarantee that a **trap** handler is always activated when a usual PLANC stack exists, it is possible for the trap handler to use the PLANC stack of the interrupted activity. DOMINOS uses this for all services which must only be called from a process and thus have a stack defined. It requires that the process has some free space on the stack.

## 5.8.5 Special rules for interrupt handlers

**Activity transparency**

When an interrupt occurs, the currently executing activity (a process or another exception handler) is suspended, and the processor starts executing the interrupt handler. After execution, the interrupted activity must be reactivated. From this activity's point of view, it **must** look as if nothing has happened. This means that the interrupt handler must not alter the context of the interrupted activity. In practice, this implies that **all** the registers used by the interrupt handler have to be temporarily stored away. The interrupt handler has no way of knowing which registers are **in use** by the interrupted activity. As already shown in the previous example, this can easily be done with two MOVEM.L assembler instructions.

**Limited services**

Since interrupts can even suspend the execution of a DOMINOS service, DOMINOS has to keep its data structures protected against corruption. This could be done by locking all data structures with a SOLO/TUTTI sequence or by using special structures. To avoid long interrupt-off times and to keep the algorithms simple this has only been done in some cases. Calling a PIR<function name> routine or TRAP #2 sequence from an interrupt handler directly or indirectly is therefore not allowed, with the following exceptions:

| | |
|---|---|
| PIRSetEvent | Set an event to a process |
| PIRFatal | Message to the outer world, that the system is going to collapse |

DOMINOS is not able to check whether or not a call for a service is invoked by an interrupt handler.

## 5.8.6 Special rules for trap handlers

Traps in this sense are **synchronous** exceptions which are generated explicitly by a process or by any other activity by using one of the following MC68K instructions:

- BKPT - Break point, used by DOMINO OPCOM
- CHK - check register against bounds
- CHK2 - check register against bounds
- cpTRAPcc - trap on coprocessor condition
- TRAP - trap unconditional
- TRAPcc - trap on condition
- TRAPV - trap on overflow

Other synchronous exceptions like bus error, address error, illegal instruction, privilege violation are not covered here. DOMINOS/OPCOM assumes that an occurrence of such a trap is not wanted and treats it as a fatal error!

**Trap handler interface**

The environment for handlers of this kind of traps is quite similar to that of interrupt handlers. The main difference is that, since the trap is programmed, the programmer may define an interface between the handler and the trap-producing activity. The programmer controls both sides of the trap. By hiding the trap-producing activity inside a routine (e.g. a library) this interface need not to be known any where else. Trap implementation in DOMINOS show this quite clearly:

**DOMINOS monitor calls**

The TRAP #2 instruction is reserved in DOMINOS for "monitor calls". The defined interface in this case is that the D0 register contains the function number and the A0 register a pointer to the appropriate parameter record. On return from the trap handler, the D0 register contains a status value.

**DOMINOS services**

It is obvious that (unlike an interrupt handler) DOMINOS need not save and restore the D0 register. The other way of calling DOMINOS (by using the routines PIR<function name>) is implemented in a similar way. However, it is defined in the specifications that no register is saved and restored, which makes this way of calling DOMINOS faster.

Also part of the interface is that the trap handler expects an existing PLANC stack (exeptions are PIRSetEvent and PIRFatal, which does not need a PLANC stack).

**DOMINOS services callable from trap handlers**

Which DOMINOS process services may be called from a trap handler depends on which environment the handler has been called from. If it was a process or a different trap handler (UDS and PME), any service may be called. If, however, it was an interrupt handler, which activated a trap handler, only PIRSetEvent and PIRFatal may be called. This to applies also nested trap handlers. The lowest level is important!

## 5.8.7 Rules for UDS and PME

**Process stack**

Inside DOMINOS, the PLANC stack of the calling process is used. Each process should therefore keep about 1/2 KByte of extra stack space plus that amount used by the PME (and even more if the UDS requires more than that).

**ERRETURN**

Use of ERRETURN inside DOMINOS is not allowed. Owing to the memory-protection system usage, the pg; error handler will reside in a memory area which has no fetch permission in supervisor mode. This results in a fatal error when ERRETURN is executed.

## 5.8.8 Implementing exception handlers

Almost all integration of UDSE with DOMINOS is done at load-time with the exception of trap and interrupt handlers. They are linked to DOMINOS at run-time. This is typically the responsibility of the **Auto_Start** process at start-up. Exception handlers different from UDS and PME are implemented to preserve compatibility with PIOCOS.

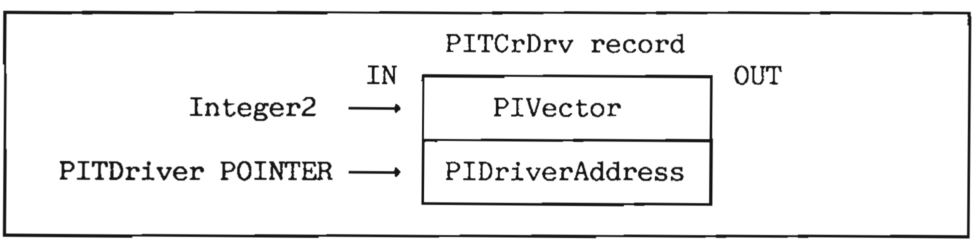
## 5.8.9 PIRCreateDriver Service

Put the address of an exception handler into the interrupt vector.

**PIVector** See the MOTOROLA manuals for vector assignment. Take also the actual DOMINO hardware implementation into consideration (e.g. interrupt levels). Some vectors are reserved for DOMINOS, and an error code is returned if they are chosen by the user. The **legal ranges** are: 3, 5:8, 10:28, 36:76, 78:255.

If the vector is in the range 64:79, then it is one of the MFP (Multi Functional Peripheral, MC68901 MFP) interrupts. In this case the corresponding channel in the MFP is enabled. Preparing timers (A and/or B) in the MFP is still the responsibility of the user program **before** calling PIRCreateDriver. Even during a power-fail restart, MFP-related drivers must be reinitialized (programming the timer(s) and calling PIRCreateDrive).

**PIDriverAddress** Pointer to the exception handler declared as ROUTINE PITDriver : <Name>.



## 5.8.10 UDSE scheduling primitives

Some internal scheduling primitives in DOMINOS are defined in the include file DOMI-UDSE-IE:IMPT, which is distributed together with DOMINOS. They can only be called from an UDSE and never from a process.

## UFindPD UDSE-primitive

Most DOMINOS services refer to a process by its process number. Internally DOMINOS refers to a process by means of a pointer to its process description (PD). This routine maps (converts) from process number to a pointer to the PD. The full content of the PD is only to be interpreted by DOMINOS, and must not be altered by a UDS. Eight INTEGER4 variables (UDSEIx, x ∊ 0:7) are free to be used by the UDSE for storing process related context.

```
ROUTINE SPECIAL INTEGER2, TPrDsPtr : UFindPD

@   Process number to find process description to.
    0 means return pointer to process description
    currently running (i.e. the process calling the UDSE).
=:  Pointer to process description TPrDs.
    NIL is returned if the process does not exist.
```

## UBlocPr UDSE-primitive

The routine **UBlocPr** blocks a process. That is, it is removed from the **ready** queue. The process **must** be in the **ready queue** when this service is requested (ready or running).

```
ROUTINE SPECIAL TPrDsPtr, INTEGER2 : UBlocPr

@    Pointer to process description
=:   Status
```

## UdeBlocPr UDSE-primitive

The routine **UDeBlocPr** de-blocks a process. That is, it is inserted into the **ready queue**. The process must be **blocked** or **dormant** when the service is called.

```
ROUTINE SPECIAL TPrDsPtr, INTEGER2 : UBlocPr

@    Pointer to process description
=:   Status
```

## 5.8.11 Implementing UDS

Up to eight User-Defined Services (UDS) can be established. They are represented in DOMINOS just like the DOMINOS services:

```
ROUTINE <option> <param. record> POINTER, INTEGER2 : <name>

 @    Pointer to parameter record (user defined)
 =:   Status (Return PIOK if call is successful)
```

Other services (PIRxxx) can be called from the UDS, but services based on current process will be done on behalf of the process calling the UDS. The UDS must be called by calling the routine PIR<name>. The UDS must be imported into the source code of the user process. Also the TRAP #2 sequence is possible but not recommended. The value UFUNCx ($x \in 0:7$) must then be loaded into the DO register prior to the call.

## 5.8.12 Implementing PME

Two groups of four routines can each be defined as process management extensions (PME):

```
   ROUTINE <option> TPrDsPtr POINTER, INTEGER2 : <name>

@    Pointer to process description
=:   Status (Return PIOK if call is successful)
```

**Start-up/ Clean-up**

Each group is triggered by a process-state transition. The first group is called each time a process is moved from the **dormant** to the **ready/running** state. The PMEs for this group typically do process start-up actions. The other group is called each time a process state is changed in the opposite direction (**running/ready/blocked** to **dormant**). This group is intended for process clean-up actions.

**Error return**

If the PME terminates (returns) with an error (Status >< PIOK), the remaining PMEs in the group are aborted, and the process does not change state.

## 5.8.13 System processes

System processes in DOMINOS exist for the sake of compatibility. In PIOCOS, the memory protection system was switched off by the scheduler as long as a system process was active. **System processes does not execute in supervisor mode!** In DOMINOS, the memory protection system is not switched off to keep the advantages. Instead, an extra window with read/write access in user mode is established in the I/O space on the MC68K bus.

**NOTE** A process should only be defined as a system process (ref. PIRCreateService), if absolutely necessary!

# Chapter 6 NUCLEUS Overview

**Usage**

NUCLEUS is intended to be used only for all Norsk Data System applications requiring fast and reliable message passing between processes within one computer. The processes may for instance be one server with several clients. NUCLEUS cannot be used for communication between computers.

**Computer**

All processes communicating via NUCLEUS must be within the same computer. By **computer** is meant one or several main CPUs and DOMINO controllers with access to the same physical memory and OCTOBUS.

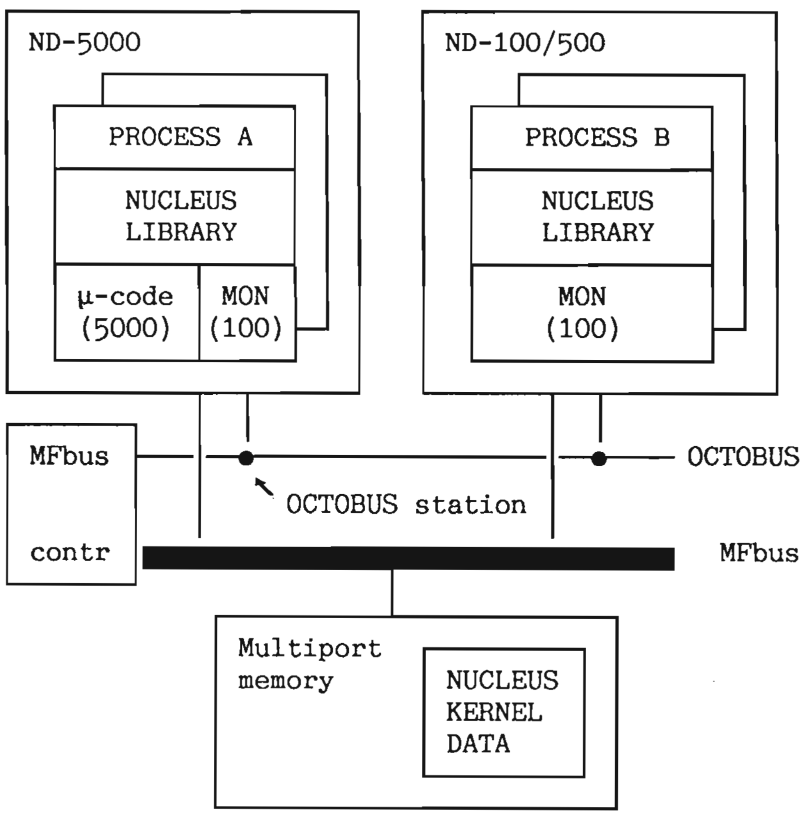



*Figure 21. Processes communicating via NUCLEUS*

**NUCLEUS Kernel**

NUCLEUS data structure reside in shared memory, operated upon by specific rules. Parts of physical memory are reserved for the data structure used by NUCLEUS.

NUCLEUS has slow and fast services. Slow services are those which not are time-critical, or are of such a nature that they need time to be carried out anyhow.

For **ND-5000**, the time-critical NUCLEUS calls nkMove, nkSend, nkReceive and nkGetInfo are microcoded to achieve required performance. All other NUCLEUS calls are executed in ND-100.

For **ND-500**, the time-critical NUCLEUS calls are not microcoded. These calls are executed in ND-100 (level 12). The NUCLEUS library in ND-500/5000 presents a standard NUCLEUS interface for applications.

**NUCLEUS library**

The services provided by NUCLEUS are independent of the CPU and operating system where the process is running.

**NUCLEUS monitor**

The NUCLEUS Monitor is a tool for inspection of tables and queues in NUCLEUS kernel.

**Communication Concepts**

Communication between processes in NUCLEUS is based on **ports** and **messages**. Their descriptions reside in physical memory shared between the CPUs. (The NUCLEUS kernel)

## 6.1 NUCLEUS library files

For manual installation of the NUCLEUS library, a diskette containing the files listed below is delivered. Choose the files needed and copy them to any user area. After loading any NUCLEUS library, a PLANC library (I-version or later) must be loaded.

| | |
|---|---|
| **NK-100-1bank-C:BRF** | NUCLEUS library for 1-bank program in ND-100. |
| **NK-100-1bank-C:BRF** | NUCLEUS library for 2-bank program in ND-100. |
| **NK-5000-C:BRF** | NUCLEUS library for ND-500/5000. |
| **NK-DOM-APPL-C:NRF** | NUCLEUS library for DOMINO Controller. |
| **NK-DOM-OS-C:NRF** | Must be loaded in DOMINO. |
| **NK-DOM-LINK-C:MODE** | Example of a DOMINO load/link job. |
| **NK-ERRCODE-C:DEFS** | Error and function codes "Constant" defs. |
| **NK-LIBRARY-C:IMPT** | Import declarations of the library routines. |

## 6.2 Including NUCLEUS in an application

| | |
|---|---|
| **NK-LIBRARY-C:IMPT** | All modules using NUCLEUS must include this file. It is common to all computers. The file contains NUCLEUS calls that can be included in a PLANC program, i.e. a library of PLANC routine calls using NUCLEUS. |

**ND-500**

The library should be loaded on a **separate segment** if the application is running on a ND-500 computer. Performance will decrease if program code and library are loaded on the same segment, because cache(in ND-500 computers) is turned **off** on the segments that libraries are loaded on.

**DOMINO**

NUCLEUS is integrated with DOMINOS on the DOMINO controller. Both NK-DOM-APPL-C:NRF and NK-DOM-OS-C:NRF must be loaded to use the NUCLEUS library inside a DOMINO controller. The mode file DOM-LINK-C:MODE contains an example of how to make a load/link job for applications using NUCLEUS inside DOMINO.

Example of linking using DOMINOS Configurator

```
@DELETE-FILE CTEST-LOAD:OUT
@(user-area)DOMI-CONFIG
CONFIG,,"TEST-LOAD:OUT"
CONFIGURATION FOR MPStdDOMINO
  LINKER is linker
  DOMAIN is test
  SYSTEM ON (user-area)DOMI-OS:NRF
  EXTENSION ON (user-area)nk-dom-os-c,(user-area)pl-mc68020
  ENDEXT
  PROCESS ON
    test,(user-area)nk-dom-appl-c,(user-area)PLANC-MC68020
  ENDPROC
  INSERT 'LIST-ENT ALL'
ENDCONFIG
```

## 6.3 Communication Concepts

Communication between processes in NUCLEUS is based on **messages** and **ports**. Their descriptions reside in physical memory shared between the CPUs (The NUCLEUS kernel).

**Message**
A message consists of a physical buffer for data and a header containing for example a buffer descriptor and link to other messages.

**Port**
A port contains for example an identification of the port owner and a pointer to received messages. Messages can be linked to a port, where they are queued in the same sequence as they arrive.

**Home port**
Every message has a home port. This is supplied when a message is created. It is used as the default port to receive a message, and is needed when a process has to answer an arbitrary process (e.g. clients & server).

**Sender port**
A message may have a sender port. This is supplied when you send the message, and is used to indicate who sent the message. Use nkGetInfo to check for who sent it. This is especially useful for servers.

**Send-reference**
In order to send a message to a port, a sendreference (to the port) must exist. The sendreference is used by NUCLEUS for access checking.

**Slow and fast services**

Creation of ports and messages are slow services, while message passing is fast. The slow functions are not needed as often as the fast ones, since the same message may be reused without being deallocated. Only the user-data need be changed between each message passing (fast services).

**Port name**

A port is uniquely identified by a symbolic port name. Processes may refer to the port by the name if they have access rights. Names cannot be abbreviated.

## 6.4 Protection in NUCLEUS

Processes are divided into two categories: system processes and public processes.

**System processes**

System processes are:

- Processes running in the DOMINO processor.
- RT-programs.
- Background programs running as user System and RT

Background programs are System processes if the user running the program originally logged in as System.

**Restrictions:** There are no restrictions for each System process. Only the total amount of resources (number of descriptors and amount of message buffer space) is limited. The amount of resources can be changed by means of the S3-configuration program (See page 181).

**Public processes**

Any process which is not a System process, is a public process.

**Restricted resources**

- **Descriptors:** For each create-port, create-message, open-port or open-return-port, a slot in the descriptor table is reserved. The number of descriptors for each public process is restricted.

- **Buffer space:** Message buffers are allocated in a common buffer pool. For each message a process creates, a fixed amount (header, fragmentation), plus the number of bytes in the create-message call, is subtracted from the allowed quota for the process.

The allowed amount of resources (number of descriptors, buffer space) common to all public processes can be set/changed on SINTRAN save areas by means of SINTRAN configuration program.

A message belongs to the user that created it. If a user creates a message, sends it away, logs out, logs in again, and the message still exists, it will still be on this users account. Public processes cannot bypass the resource restrictions by logging out and in again.

If someone tries to return a message to a home port that does not exist any more (the user may have logged out), the message will be deleted, and subtracted from the users account.

**Naming:**

- Only system processes can create names.
- Processes which do not have access rights to a named port cannot open a sendreference to it. Access rights are determined by the access parameter in the create-port-name call.
- Only the owner of a port can delete the port's name.
- The "name" is a string of 32 bytes.
- Any combination of alphanumeric characters is allowed as a port name. For instance "NIL" is a legal name.
- One port can be given several names.
- Ports must have different names.

## 6.5 Configuration of NUCLEUS

The standard NUCLEUS configuration is defined when SINTRAN is generated. Changes in the NUCLEUS configuration can be made by means of a new function in the SINTRAN monitor call MON CONFIG. The SINTRAN configuration program is updated to handle reconfiguration of NUCLEUS.

**Configuration parameters**

Number of descriptors = number of ports and messages.
Buffer space = space used for messages.
**Default values in the table may have been changed.**

| NUCLEUS command parameters in S3-CONFIGURATION program | Default values for NUCLEUS |
|---|---|
| Message buffer space for system processes in pages | 250 Kbytes [1]) (125 pages) |
| Number of descriptors for all system processes | 500 [1]) |
| Message buffer space for all public processes in pages | 250 Kbytes [2]) (125 pages) |
| Number of descriptors for all public processes | 300 [2]) |
| Message buffer space per public process in pages | 10 Kbytes (5 pages) |
| Number of descriptors per public process | 10 |
| Trace buffer space in pages | 2 Kbytes (1 page) |

See notes on next page.

[1]) Assuming 1 disk DOMINO, 8 databases, 16 socket channels.
[2]) Assuming 2.5 Kbytes, 3 descriptors per access library, 100 public processes.

During start-up, NUCLEUS allocates first available memory in multiport memory.

## 6.6 NUCLEUS in ND-100

NUCLEUS in ND-100 consists of code on SINTRAN page tables MPIT, DPIT, RPIT and COMMON area. In addition, the NUCLEUS server executes as an RT program on SINTRAN page table SPIT. The NUCLEUS name server executes as an RT program on user page tables. Both servers are integrated with SINTRAN. During start-up of SINTRAN, the servers are started by SINTRAN itself.

## 6.7 NUCLEUS in DOMINO Controller

Starting NUCLEUS in DOMINO is invinsible for applications. NUCLEUS in the DOMINO Controller is able to handle processes with different levels of priority.

# Chapter 7 NUCLEUS library

This chapter describes the routine calls available from NUCLEUS.

## 7.1 Summary of NUCLEUS calls

**NOTE:**

> In calls with only one function, the function value must be zero.

**CREATE PORT** nkCrePort(function,events,=port)

function = 0 ; **nkfNoDelayAbort**
function = 1 ; **nkfDelayAbort**

**CREATE NAME** nkCreName(function,access,name,port)

**OPEN PORT** nkOpenPort(function,name,=sendreference)

**OPEN RETURN PORT** nkOpenReturnPort(function,message,=sendreference)

function = 0 ; **nkfOpenHomePort**
function = 1 ; **nkfOpenLastPort**

**DELETE NAME** nkDelName(function,name,port)

**CREATE MESSAGE** nkCreMessage(function,bytes,homeport,=message)

**MOVE** nkMove(function,message,displacement,(=)data,
=bytes)

function = 0 ; **nkfRead**
function = 1 ; **nkfWrite**
function = 2 ; **nkfInsert**

**SEND** nkSend(function,port,sendreference,message)

**RECEIVE** nkReceive(function,port,=message,=bytes)

**CLOSE** nkClose(function,port or message or sendreference)

function = 0 ; Port or sendreference

function = 0 ; **nkfRemove**
function = 1 ; **nkfReject**
Only for messages

**GET INFO** nkGetInfo(function,port or message or
sendreference,=value(bytes pointer))

function = 0 ; **nkfSize**
function = 1 ; **nkfLength**
function = 2 ; **nkfHomeid**
function = 3 ; **nkfLastid**
function = 4 ; **nkfBuffer**
function = 5 ; **nkfQueue**

**GET INFO** nkVersion(function,<station no,=version)

function = 0 ; **nkfLibrary**
function = 1 ; **nkfKernel**
function = 2 ; **nkfStation**

## 7.2 Parameters in NUCLEUS calls

The status from a NUCLEUS call is returned as an outvalue. (Always INTEGER4)

The first parameter is a function number. In calls with only one function, the function value must be zero. Five NUCLEUS calls have more than one function. To specify the function in a call, you may use either the function number or a symbolic subfunction name.

| NUCLEUS call | Function number | Subfunction name |
|---|---|---|
| **nkCrePort** | 0 | nkfNoDelayAbort |
| | 1 | nkfDelayAbort |
| **nkOpenReturnPort** | 0 | nkfOpenHomePort |
| | 1 | nkfOpenLastPort |
| **nkGetInfo** | 0 | nkfSize |
| | 1 | nkfLength |
| | 2 | nkfHomeid |
| | 3 | nkfLastid |
| | 4 | nkfBuffer |
| | 5 | nkfQueue |
| **nkMove** | 0 | nkfRead |
| | 1 | nkfWrite |
| | 2 | nkfInsert |
| **nkClose**[1] | 0 | nkfRemove |
| | 1 | nkfReject |
| **nkVersion** | 0 | nkfLibrary |
| | 1 | nkfKernel |
| | 2 | nkfStation |

[1]) Subfunction names are valid for **messages** only.

*Table 5. Function numbers and names in NUCLEUS calls*

## 7.2.1 NUCLEUS status codes

**Error codes**

NUCLEUS operation error/status codes are found in Appendix C, on page 271-272.

The following status codes may be returned after a service. The constants denoting the status codes are in the include file NK-ERRCODE:DEFS

| Constant | Octal val | Meaning |
|---|---|---|
| nke_ERROR_BASE | 101000b | Base number for Nucleus errors |
| nke_ILLPAR | 101001b | Invalid parameter value |
| nke_ILLTYPE | 101002b | Wrong type used,- port, message or send reference |
| nke_NOMESS | 101003b | Both port and message in Send reference may not be zero |
| nke_ILLNO | 101004b | Port, message or send reference outside range |
| nke_NOTLOCAL | 101005b | Receive from remote port |
| nke_OUTSIDE | 101006b | Displacement outside buffer |
| nke_DESCARRFULL | 101007b | Descriptor table full |
| nke_BUFFULL | 101010b | Message buffer area full |
| nke_NAMEFULL | 101011b | Name table full |
| nke_NAMENOTFOUND | 101012b | Port name not defined |
| nke_NAMEUSED | 101013b | Port name already defined |
| nke_NOACCESS | 101014b | No access to given port, message or send reference |
| nke_ILLNETADDRESS | 101015b | Net address not found |
| nke_ILLKERNELNO | 101016b | Invalid kernel number |
| nke_NETTABFULL | 101017b | Net table full |
| nke_PROTOCERROR | 101020b | Inconsistent Nucleus module versions installed |
| nke_REJECTED | 101021b | Message rejected by receive process |

Continue on next page...

| Constant | Octal val | Explanation |
|---|---|---|
| nke_PORTNOTFOUND | 101022b | Port reference not defined in name server |
| nke_LOCK | 101023b | Unable to lock port |
| nke_NOTEVENBYTE | 101024b | Displacement not on even byte (only for ND-100) |
| nke_NOTINITIALISED | 101025b | Nucleus not started |
| nke_NAMEPORTUSED | 101026b | The Nameserver port is already initialised |
| nke_NAMEINDEXERROR | 101027b | Index error in Nameserver request |
| nke_INCONSISTENT | 101030b | Inconsistent structure in name server |
| nke_TOOMANYBYTES | 101031b | Buffer provided is too small |
| nke_PORTCLOSED | 101032b | Receive port is closed. |
| nke_ILLFUNC | 101033b | Invalid Function code |
| nke_PROTECTED | 101034b | Attempt to use protected Function |
| nke_ILLHARDWARE | 101035b | Not correct hardware configuration |
| nke_FATAL | 101036b | Fatal error in Nucleus |
| nke_QTABFULL | 101037b | Too many concurrent Nucleus users (quota table full) |
| nke_QUOTAUSED | 101040b | No more Nucleus resources available for this user |
| nke_ILLUSER | 101041b | Unknown user area identifier |
| nke_KICKLOCK | 101042b | Timeout when waiting for lock (kick-queue) |
| nke_DELAYTABFULL | 101043b | Unable to create more ports using delayed abort |
| nke_NOTAVAILABLE | 101044b | NUCLEUS not available in CPU. (not started or stopped) |
| nke_ILLVERSION | 101045b | Invalid version of NUCLEUS library |

*Table 6. NUCLEUS status/error codes*

| | nkCrePort | nkCreName | nkOpenPort | nkOpenReturnPort | nkDelName | nkCreMessage | nkMove | nkSend | nkReceive | nkClose | nkGetInfo | nkVersion |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| nke_BUFFULL | | | | | | X | | | | | | |
| nke_DELAYTABFULL | X | | | | | | | | | | | |
| nke_DESCARRFULL | X | | | | | X | | | | | | |
| nke_FATAL | X | X | X | X | X | X | | | | X | | |
| nke_ILLFUNC | X | X | X | X | X | X | X | X | X | X | X | X |
| nke_ILLHARDWARE | | | | | | | | | | | | |
| nke_ILLKERNELNO | | | | | | | | | | | | |
| nke_ILLNETADDRESS | | | | | | | | | | | | |
| nke_ILLNO | | X | X | X | X | X | X | X | X | X | X | |
| nke_ILLPAR | | X | | | | | | | | | | X |
| nke_ILLTYPE | | X | | X | X | X | X | X | X | | X | |
| nke_ILLUSER | X | | | | | X | | | | | | |
| nke_ILLVERSION | X | X | X | X | X | X | X | X | X | X | X | X |
| nke_INCONSISTENT | | | | | | | | | | | | |
| nke_KICKLOCK | | | | | | | | | | | | |
| nke_LOCK | | | | | | | | | | | | |
| nke_NAMEFULL | | X | | | | | | | | | | |
| nke_NAMEINDEXERROR | | | | | | | | | | | | |
| nke_NAMENOTFOUND | | | X | | X | | | | | | | |
| nke_NAMEPORTUSED | | | | | | | | | | | | |
| nke_NAMEUSED | | X | | | | | | | | | | |
| nke_NETTABFULL | | | | | | | | | | | | |
| nke_NOACCESS | | | X | X | | X | X | X | X | X | X | |
| nke_NOMESS | | | | | | | | X | | | | |
| nke_NOTAVAILABLE | | | | | | | | | | | | X |
| nke_NOTEVENBYTE | | | | | | | X | | | | | |
| nke_NOTINITIALISED | X | X | X | X | X | X | X | X | X | X | X | |
| nke_NOTLOCAL | | | | | | | | | | | | |
| nke_OUTSIDE | | | | | | X | X | | | | | |
| nke_PORTCLOSED | | | | | | | | X | | | | |
| nke_PORTNOTFOUND | | | | | | | | | | | | |
| nke_PROTECTED | | X | | | | | | | | | | |
| nke_PROTOCERROR | | | | | | | | | | | | |
| nke_QTABFULL | | | | | | | | | | | | |
| nke_QUOTAUSED | X | | | | | X | | | | | | |
| nke_REJECTED | | | | | | | | | X | | | |
| nke_TOOMANYBYTES | | | | | | | | | | | X | |

*Table 7. NUCLEUS calls and error/status codes*

## 7.3 NUCLEUS Call Interface

Every PLANC routine call has an outvalue, but no invalue, i.e.:

ROUTINE VOID,INTEGER4(....

### 7.3.1 Create port

**Purpose** Create a new port. The creating process becomes the port owner.

**Syntax** **nkCrePort**(<function>,<events>,<=port>)

**Parameter description**

<function> = 0 Abort not delayed. **nkfNoDelayAbort**
= 1 Delay abort. **nkfDelayAbort**
For further information about nkfDelayAbort, see next page.

<events> ≠ 0 If ND-100 or ND-500: The process will be activated when the first message arrives at the empty port.
ND-100: Process is stopped by MON 267 (TimeOut).
ND-500: Process is stopped by MON 501 (StopProcess) or MON 514 (ND500TimeOut).
If DOMINO: Events will be used together with the event system in DOMINOS.
Event bit 30 and 31 are used by NUCLEUS itself. These bits cannot be used by any application!
= 0 The process will not be activated.

<=port> = Port number.

**Rules** The subfunction **nkfDelayAbort** (function=1) is yet only available for ND-5000 System processes

**PLANC routine call**

```
ROUTINE VOID, INTEGER4         & status
            ( INTEGER4,        & function
              INTEGER4,        & events
              INTEGER4 WRITE)  & port
             : nkCrePort
```

**ERROR CODES**

```
nke_DELAYTABFULL       % No more space in delay abort
                         table
nke_DESCARRFULL        % Descriptor table full
nke_FATAL              % Fatal error in NUCLEUS
nke_ILLFUNC            % Invalid function code
nke_ILLUSER            % Unknown user identifier
                         (fatal error)
nke_ILLVERSION         % Invalid version of NUCLEUS
                         library.
nke_NOTINITIALISED     % NUCLEUS not started
nke_QUOTAUSED          % Quota exceeded for this user
```

## 7.3.1.1 Delayed abort for NUCLEUS

If a port is created with the subfunction nkfDelayAbort, then the process that owns the port will be delayed aborted (hang in abortion state) until all messages with this port defined as home port are returned to the home port.

**NOTE !** This subfunction is yet only available for ND-5000 System processes.

**Example** In some cases DOMINO operates directly on fixed segments of ND-5000 processes. It is important that the process is not aborted (and the segments unfixed) while DOMINO carries out data transfers. To avoid this, process abortion should be delayed while data transfer control messages still remain.

To solve the problem of unwanted abortion of a process, ports that are home ports for data transfer control messages should use the subfunction **nkfDelayAbort** when they are created.

```
nkCrePort(nkfDelayAbort,....
```

## 7.3.2 Create port name

**Purpose**

Assign a name to a port, so that other processes can refer to it.

**Syntax**

**nkCreName**(<function>,<access>,<name>,<port>)

**Parameter description**

<function> = 0

<access> = 0 Only System processes have access to this port.
= 1 System and public access.

<name> = Symbolic name of port.

<port> = Number of port to be assigned a name.

**Rules**

1. The call is allowed for System processes only.
2. Only the owner of the port is allowed to use this call.
3. One port may have several names.
4. The "name" is a string of 32 bytes.
5. Any combination of alphanumeric characters is allowed as a port name.
6. Different ports cannot have equal names.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4       & status
            ( INTEGER4,      & function
              INTEGER4,      & access
              BYTES POINTER,& name
              INTEGER4 )     & port
            : nkCreName
```

```
ERROR CODES     nke_FATAL                % Fatal error in NUCLEUS
                nke_ILLFUNC              % Invalid function code
                nke_ILLNO                % Invalid descriptor number
                nke_ILLPAR               % Invalid parameter value
                                           ( access type ≠ 0 , 1)
                nke_ILLTYPE              % Invalid descriptor type
                nke_ILLVERSION           % Invalid version of NUCLEUS
                                           library.
                nke_NAMEFULL             % Name table full
                nke_NAMEUSED             % Name already used
                nke_NOTINITIALISED       % NUCLEUS not started
                nke_PROTECTED            % Function is protected
```

## 7.3.3 Open port

**Purpose** This service will be used to get a send reference to a named port.

**Syntax** **nkOpenPort**(<function>,<name>,<=sendreference>)

**Parameter description**

<function> = 0

<name> = Symbolic name of port.

<=sendreference> = Sendreference number to port.

**Rules**

1. A public process can open a port only if access to the port (set in nkCreName call) is allowed both for System and public processes.
2. Processes using this call must know the name of the port.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4          & status
            ( INTEGER4,         & function
              BYTES POINTER,    & name
              INTEGER4 WRITE)   & sendreference
             : nkOpenPort
```

**ERROR CODES**

| | |
|---|---|
| nke_FATAL | % Fatal error in NUCLEUS |
| nke_ILLFUNC | % Invalid function code |
| nke_ILLNO | % Invalid descriptor number |
| nke_ILLVERSION | % Invalid version of NUCLEUS library. |
| nke_NAMENOTFOUND | % Name not in name table |
| nke_NOACCESS | % Not access to port |
| nke_NOTINITIALISED | % NUCLEUS not started |

## 7.3.4 Open return port

**Purpose**

Open a send reference to the home port or last sender port of a message.

**Syntax**

**nkOpenReturnPort**(<function>,<message>,
<=sendreference>)

**Parameter description**

<function> = 0 Reference to the home port of the message.
= 1 Reference to the last port the message was sent from.

<message> = Message number.

<=sendreference>= Send reference to home port or last sender port.

**Rules**

1. Only the owner of the message is allowed to use this call.

A "receive" on a message, implies that owner is set. A message that is sent, but not received, has no owner.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4        & status
            ( INTEGER4,       & function
              INTEGER4,       & message
              INTEGER4 WRITE) & sendreference
            : nkOpenReturnPort
```

**ERROR CODES**

```
nke_FATAL              % Fatal error in NUCLEUS
nke_ILLFUNC            % Invalid function code
nke_ILLNO              % Invalid descriptor number
nke_ILLTYPE            % Invalid descriptor type
nke_ILLVERSION         % Invalid version of NUCLEUS
                         library.
nke_NOACCESS           % Not access to port
nke_NOTINITIALISED     % NUCLEUS not started
nke_PORTNOTFOUND       % Port not found in name server
```

## 7.3.5 Delete port name

**Purpose**

Delete the symbolic name of a port. The port itself is not removed.

**Syntax**

**nkDelName**(<function>,<name>,<port>)

**Parameter description**

<function> = 0

<name> = Symbolic name of the port.

<port> = Number of the corresponding port.

**Rules**

The symbolic name of a port can only be deleted by the owner of the port. Correspondence between port name and port number is checked.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4        & status
            ( INTEGER4,       & function
              BYTES POINTER,  & name
              INTEGER4)       & port
            : nkDelName
```

**ERROR CODES**

```
nke_FATAL              % Fatal error in NUCLEUS
nke_ILLFUNC            % Invalid function code
nke_ILLNO              % Invalid descriptor number
nke_ILLTYPE            % Invalid descriptor type
nke_ILLVERSION         % Invalid version of NUCLEUS
                         library.
nke_NAMENOTFOUND       % Name not in name table
nke_NOTINITIALISED     % NUCLEUS not started
```

## 7.3.6 Create message

**Purpose**

Allocate a message buffer in a contiguous area of physical memory. It can be written into and read from, using the fast services **nkMove**

The creating process owns and has exclusive access to the message until it is sent to a port. The access to the message is lost when it is sent to another process.

The homeport must be a port owned by the creating process. Zero may be supplied to indicate dummy home port, meaning that the message will be lost and deallocated if it is sent to the home port.

**Syntax**

**nkCreMessage**(<function>,<bytes>,<homeport>,
<=message>)

**Parameter description**

<function> = 0

<bytes> = Max. number of bytes in the message.

<homeport> = Home port number.

<=message> = Message number.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4          & status
            ( INTEGER4,         & function
              INTEGER4,         & bytes
              INTEGER4,         & homeport
              INTEGER4 WRITE)   & message
            : nkCreMessage
```

**ERROR CODES**

```
nke_BUFFULL            % Buffer area full
nke_DESCARRFULL        % Descriptor table full
nke_FATAL              % Fatal error in NUCLEUS
nke_ILLFUNC            % Invalid function code
nke_ILLNO              % Invalid descriptor number
nke_ILLTYPE            % Invalid descriptor type
nke_ILLUSER            % Unknown user identifier
nke_ILLVERSION         % Invalid version of NUCLEUS
                         library.
nke_NOACCESS           % Not access to port
nke_NOTINITIALISED     % NUCLEUS not started
nke_OUTSIDE            % Displacement outside buffer
nke_QUOTAUSED          % Quota exceeded for this user
```

## 7.3.7 Read or write a message

**Purpose**

**Write** user data into the message buffer of a message from index <mesdispl> and upwards. The write operation terminates either when all user data is written, or when the message buffer becomes full.

**Read** data from the message buffer, starting from the message displacement. The reading terminates either when the whole message has been read, or when the user data area becomes full.

**Syntax**

**nkMove**(<function>,<message>,<displacement>,
<(=)data>,<=bytes>)

**Parameter description**

<function> = 0 => Read message. **nkfRead.**
= 1 => Write message. **nkfWrite.**
= 2 => Insert. Same function as Write, but the byte pointer is not set if the message is smaller then the old message. **nkfInsert.**

<message> = Number of the message to be read/written.

<displacement> = Displacement within message buffer.

<(=)data> = User data to be read/written.

<=bytes> = Number of bytes actually read/written

**Rules**

1. The message buffer is identical to the declaration: Bytes : message(0:msglngth-1).

2. In the ND-100 maxindex and minindex in the byte pointer must be in the range 0-64511. Displacement must be an even number for ND-100.

3. "NIL" cannot be used as an empty message. An empty message can be specified as an empty byte string, i.e. : ADDR ' '
   Bytes pointer with minindex= 0 and maxindex=-1 is also an empty message.

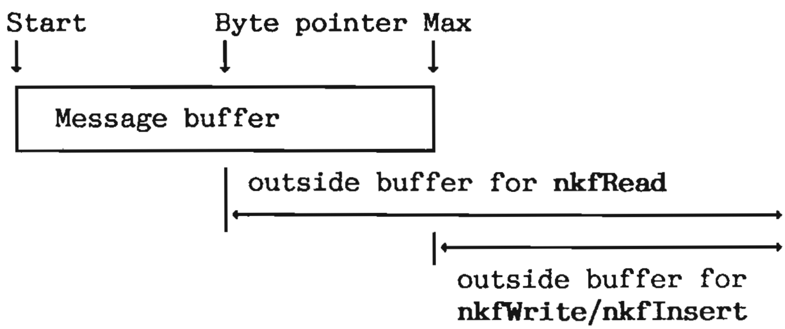
**PLANC routine call**

```
ROUTINE VOID, INTEGER4          & status
            ( INTEGER4,         & function
              INTEGER4,         & message
              INTEGER4,         & displacement
              BYTES POINTER     & data
              INTEGER4 WRITE)   & bytes
            : nkMove
```

**ERROR CODES**

| | |
|---|---|
| nke_ILLFUNC | % Invalid function code |
| nke_ILLNO | % Invalid descriptor number |
| nke_ILLTYPE | % Invalid descriptor type |
| nke_ILLVERSION | % Invalid version of NUCLEUS library. |
| nke_NOACCESS | % Not access to port |
| nke_NOTEVENBYTE | % Displacement not even byte. (Only returned for ND-100) |
| nke_NOTINITIALISED | % NUCLEUS not started |
| nke_OUTSIDE | % Displacement outside buffer |

**write/insert**: outside max buffer.
**read**: outside current byte counter.



## 7.3.8 Send message

**Purpose**

Send a message to a port, provided that the sending process has access to the message. The process loses its access to this message. The message is appended at the end of the message queue at the destination port.

If the queue at the destination port is empty, then the message will activate the process which created the destination port, if so specified at create time.

**Syntax**

**nkSend**(<function>,<port>,<sendref.>,<message>)

**Parameter description**

<function> = 0

<port> = Port number to identify who sent the message(Last port). New sender port is not set if the port number equals zero.

<sendref.> = Sendreference to port to receive the message.
If sendreference = 0, the message is sent to the home port of the message.

<message> = Message number of the message to be sent. If the message number is equal to zero this call will not send a message, but perform a restart of the process of the destination port.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4     & status
            ( INTEGER4,    & function
              INTEGER4,    & port
              INTEGER4,    & sendreference
              INTEGER4)    & message
            : nkSend
```

**ERROR CODES**

| | |
|---|---|
| nke_ILLFUNC | % Invalid function code. |
| nke_ILLNO | % Invalid descriptor number. |
| nke_ILLTYPE | % Invalid descriptor type. |
| nke_ILLVERSION | % Invalid version of NUCLEUS library. |
| nke_NOACCESS | % Not access to port. |
| nke_NOMESS | % No port and no message in SEND. |
| nke_NOTINITIALISED | % NUCLEUS not started. |
| nke_PORTCLOSED | % Receive port is closed. |

Status **nke_PortClosed** is returned if the port to receive the message is closed. If sendreference is not specified (send message to home port) and the home port is closed, then the message is deallocated.

If status **nke_PortClosed** is returned and send-reference is specified, then the sendreference should be closed. This sendreference is no longer valid because the port to receive the message is closed.

## 7.3.9 Receive message

**Purpose**

The first message in the queue is received. If the queue is empty, message number zero is returned. The receiving process gets access to the message, and may read from and write to it.

**Syntax**

**nkReceive**(<function>,<port>,<=message>,<=bytes>)

**Parameter description**

<function> = 0

<port> = Port number. Identifies the port from which the message will be received.

<=message> = Message number.

<=bytes> = Number of bytes written into the message buffer by the sending process It is equal or less than the message size. You can use **nkGetInfo** to get the message size and who sent it.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4         & status
            ( INTEGER4,        & function
              INTEGER4,        & port
              INTEGER4 WRITE,  & message
              INTEGER4 WRITE)  & bytes
            : nkReceive
```

**ERROR CODES**

| | |
|---|---|
| nke_ILLFUNC | % Invalid function code |
| nke_ILLNO | % Invalid descriptor number |
| nke_ILLTYPE | % Invalid descriptor type |
| nke_ILLVERSION | % Invalid version of NUCLEUS library. |
| nke_NOACCESS | % Not access to port |
| nke_NOTINITIALISED | % NUCLEUS not started |
| nke_REJECTED | % Return to sender |

## 7.3.10 Get Info

**Purpose**

Get information on the specified message or port.

**Syntax**

**nkGetInfo**(<function>,<message or port or sendreference>,<=value>)

**Parameter description**

<function> = 0 : **nkfSize**. Maximum message size.
= 1 : **nkfLength**. Used message length.
= 2 : **nkfHomeid**.
If message: Home port identifier.
If port: Port identifier.
If send reference: Destination port identifier.
= 3 : **nkfLastid**. Identifies the last port that sent this message.
= 4 : **nkfBuffer**. Buffer address of the message in NUCLEUS kernel.
= 5 : **nkfQueue**.
0 => port has no message.
1 => port has one or more messages.

<function> = 0, 1, 4 , 5 returns 32 bits(4 bytes).

<function> = 2 and 3 returns 64 bits (8 bytes). For future NUCLEUS extension, all applications must be prepared for returning 128 bits (16 bytes).

<function> = 0, 1, 3, 4 can be used for **messages** only.

<function> = 5 can be used for **ports** only. Returned as INTEGER4.

> **NOTE !**
> If <function> = 2 or 3, the identifiers returned can only be used to compare other identifiers returned from nkGetInfo.
> Do not extract any other information.

**Parameter description**

<message = Message number.
or port or = Port number.
sendreference> = Sendreference number.

<=value> = Message, port or sendreference information.

**Rules**

Only the process having access to the message, port or sendreference is allowed to use this call.

**PLANC routine call**

```
ROUTINE VOID, INTEGER4       & status
            ( INTEGER4,      & function
              INTEGER4,      & message, port or sendreference
              BYTES POINTER)& value
            : nkGetInfo
```

**ERROR CODES**

| | |
|---|---|
| nke_ILLFUNC | % Invalid function code |
| nke_ILLNO | % Invalid descriptor number |
| nke_ILLTYPE | % Invalid descriptor type |
| nke_ILLVERSION | % Invalid version of NUCLEUS library. |
| nke_NOACCESS | % Not access to port |
| nke_NOTINITIALISED | % NUCLEUS not started |
| nke_TOOMANYBYTES | % Too many bytes (maxindex or minindex outside limits) |

## 7.3.11 Close port, message or sendreference

**Purpose**

Close a port, message or sendreference.

**Closing a message**

If function (see next page) = 0 (nkfRemove), then the message is deallocated.
If function = 1 (nkfReject), then the message is closed according to the following algorithm:

```
IF lastport in message is set and not closed THEN
  IF lastport owned by invoking process THEN
     deallocate message
   ELSE
     send message to lastport with status rejected
  ENDIF
ELSE
  IF homeport closed or owned by invoking process
  THEN deallocate message
  ELSE
     send message to homeport with status rejected
  ENDIF
ENDIF
```

**Closing a port** results in deletion of the port number and all of the ports symbolic names.
If there exists messages (in queue to the port) that the port has not yet received, the messages will be closed according to the algorithm above (function = 1 [**nkfReject**]).

**Closing a send reference.** The send reference is closed.

> NOTE !
> When a process is aborted or a CPU in the system is rebooted, messages are deallocated/closed as in function=1 (see next page).

**Syntax**

**nkClose**(<function>,<port or message or sendref.>)

**Parameter description**

**<function>** = 0

<port = Port number to be closed. If the port is named, all names defined with the call nkCreName will be removed.
or
message = Message number to be **deallocated.** (**nkfRemove**)
or
sendref.> = Sendreference to be closed.

**<function>** = 1 **[nkfReject]** The **message** is closed according to the algorithm for closing a message.

**nkClose(0,-1)** will close all ports, sendreferences and deallocate all messages owned by the process.

**nkClose(1,-1)** will close all ports, sendreferences and messages owned by the process.

**Rules**

1. A message can only be closed by the process that currently has access to the message.
2. Port or sendreference can only be closed by the process which owns the port/sendreference

**PLANC routine call**

```
ROUTINE VOID, INTEGER4  & status
          ( INTEGER4, & function
            INTEGER4) & message or port or sendreference
          : nkClose
```

**ERROR CODES**

| | |
|---|---|
| nke_FATAL | % Fatal error in NUCLEUS |
| nke_ILLFUNC | % Invalid function code |
| nke_ILLNO | % Invalid descriptor number |
| nke_ILLVERSION | % Invalid version of NUCLEUS library. |
| nke_NOACCESS | % Not access to port |
| nke_NOTINITIALISED | % NUCLEUS not started |

## 7.3.12 Get Version

**Purpose** Get version of different NUCLEUS parts. May be useful for version control.

**Syntax** **nkVersion**(<function>,<station no>,<=version>)

**Parameter description**

<function> = 0 => Version of NUCLEUS library (NUCLEUS library application is linked to). [**nkfLibrary**]
= 1 => Version of NUCLEUS kernel data layout. [**nkfKernel**]
= 2 => Version of NUCLEUS last loaded in <station no>. [**nkfStation**]

<station no> = Octobus station number.

<=version> = Version consisting of three alphanumeric characters.

**Rules**

1. The parameter <station no> must be in range 1 to 77B and is valid only for function 2.
2. The parameter <version> is yet only returned for Domino controllers.

**Planc routine call**

```
ROUTINE VOID, INTERGER4          &   status
            ( INTERGER4,         &   function
              INTERGER4,         &   station no
              BYTES POINTER)     &   version
            : nkVersion)
```

**ERROR CODES**

nke_ILLFUNC % Invalid function code.
nke_ILLPAR % version string too small,
% or cluster id outside range.
nke_ILLVERSION % Invalid version of NUCLEUS library.
nke_NOTAVAILABLE % Domino contr. not yet started.

## 7.4 Brief introduction to tables in NUCLEUS kernel

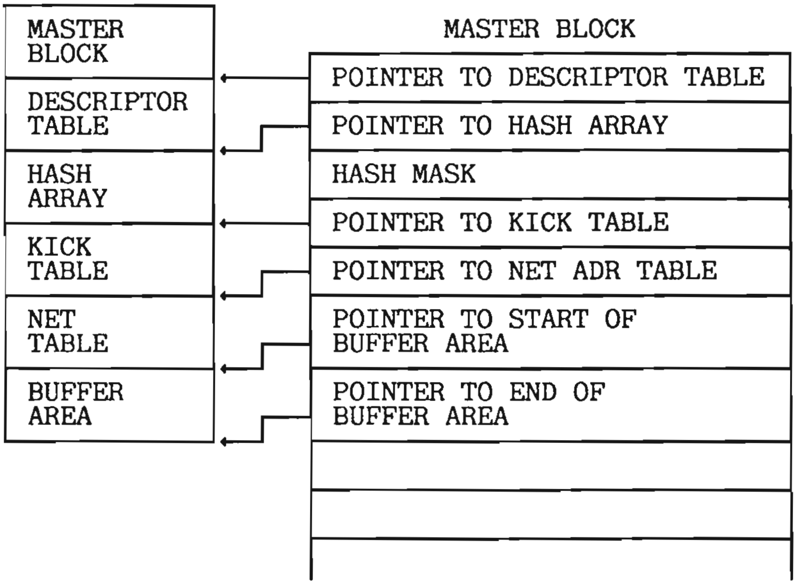



*Figure 23. Tables in NUCLEUS kernel*

| | | |
|---|---|---|
| HEAD | LOCK | Used for TSET |
| | 2 | TYPE= 2 => Message |
| | OWNER | Owner of message. Used for access check |
| | FREELINK | Link in freelist |
| | USER | Used for quota control |
| LINK | | From (receiving) port |
| BUFFERPOINTER | | Pointer to buffer record |
| HOMEPORT | | Pointer to (home)port |
| HASHLINK | | Identifying messages from remote |
| COMSTAT | | Shadow message usage |
| OWNINDEX | | Descriptor number |
| TRACECOND | | 0 = NO trace |

*Figure 24. Record layout for a message in descriptor table*

| | | | |
|---|---|---|---|
| HEAD | LOCK | → for TSET | |
| | 3 | → TYPE= 3 => Port | |
| | OWNER | → Owner of port. Used for access check | |
| | FREELINK | → Link in freelist | |
| | USER | → Used for quota control | |
| | MESS HEAD | → Start of message queue | |
| | MESS TAIL | → End of message queue | |
| KICKLINK | | → Link kicked ports together | |
| | KICK HEAD | → Points to queue head in kicktable | |
| | KICK DEST. | → Index in kicktable(=OCTOBUS station no) | |
| | INQUEUE | → = 0 => NOT in kickqueue | |
| | KICK PROC. | → Process to be kicked | |
| | EVENTS 1 | → Events to be set | |
| | EVENTS 2 | → More event info | |
| OWNINDEX | | → Descriptor number | Identifies this port. ( OWNID ) |
| PRANDOM | | → Magic number | |
| NETTADDRESS | | → To complete identifier | |
| OPENCOUNT | | → Number of times opened for receive | |
| NAMED | | → Number of names for this port | |

*Figure 25. Record layout for a homeport in descriptor table*

| | | | |
|---|---|---|---|
| HEAD | LOCK | | |
| | 4 | → | TYPE= 4 => Sendreference |
| | OWNER | → | Owner of sendref. Used for access check |
| | FREELINK | → | Link in freelist |
| | USER | → | Used for quota control |
| DEST.PORT ID. | | → | ID to receiving port |
| DESTINATIONPORT | | → | Destination pointer |

*Figure 26. Record layout for a sendref in descriptor table*

| | | |
|---|---|---|
| PROTOCOL | → | NUCLEUS protocol version |
| MESSAGE STATUS | → | Message status - e.g. rejected. |
| PORT ID | → | Destination port |
| MESSAGE ID | → | Original message identifier |
| HOME ID | → | Home receive port |
| LAST ID | → | Last send port |
| SIZE | → | Maximum number of bytes |
| LENGTH | → | Bytes used |
| BUFFER(0:(-1)) | → | Start data buffer |

*Figure 27. Message buffer layout in bufferarea*

## 7.4.1 NUCLEUS call sequence - an example

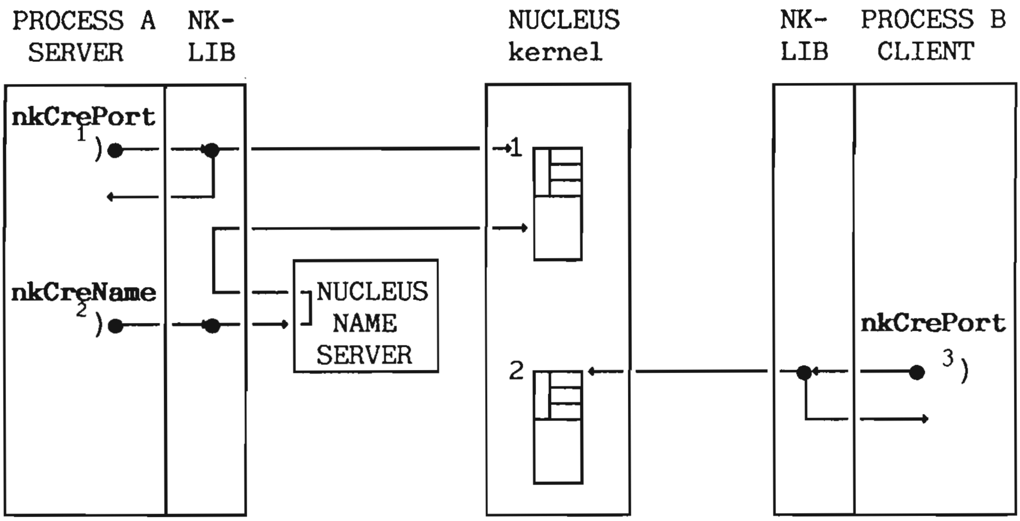
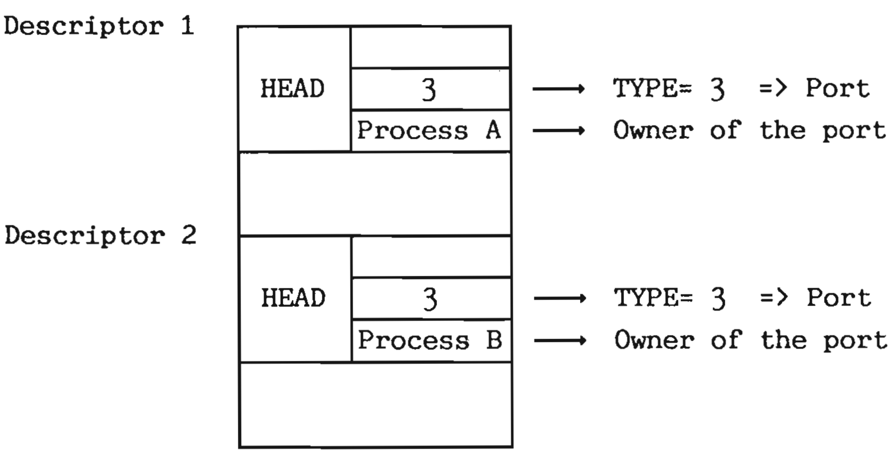



*Figure 28. Creating ports and names in NUCLEUS*

1) Process A (server) creates a port (nkCrePort), and
2) assigns a name to it (nkCreName).
3) Process B (client) creates a port (nkCrePort).




The two nkCrePort calls each reserves a descriptor in the NUCLEUS descriptor table. Nucleus name server checks that the name assigned to the port by Process A (nkCreName)is unique. The port name (ownid) in descriptor 1 is updated.

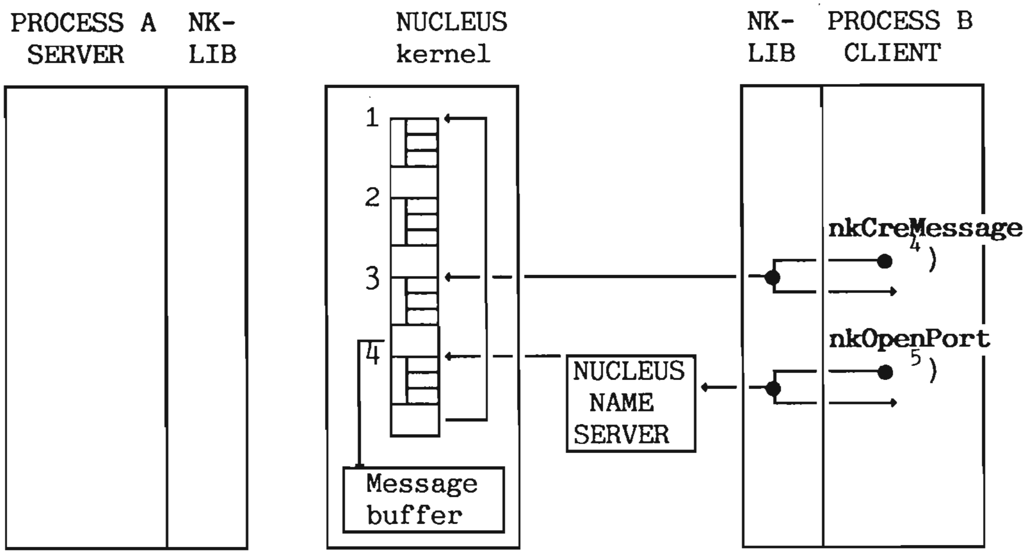
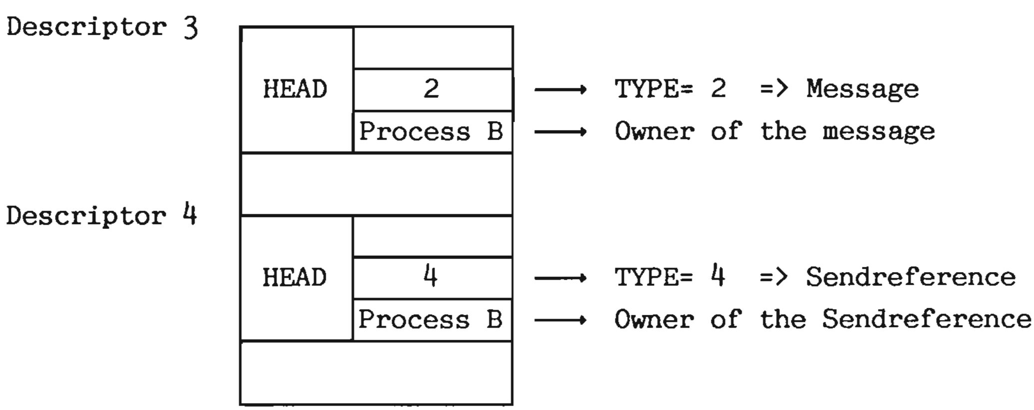



*Figure 29. Create message and open port*

4) Process B (client) creates a message (nkCreMessage)
5) Process B (client) opens the port created by Process A, to get a sendreference to the port.




Another two descriptors in the NUCLEUS descriptor table are reserved by the calls nkCreMessage and nkOpenPort. Message buffer is allocated in buffer area. It is checked that Process B has access to the port. The categories of processes that may open a sendreference to the port are given by the owner of the port (Process A), by means of the nkCreName call. The sendreference descriptor has a pointer to receiving port #1.

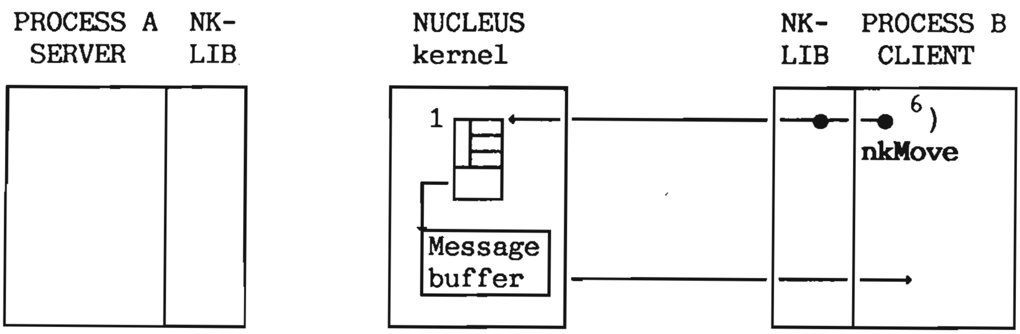



*Figure 30. Write a message into the message buffer*

[6]) The function nkfWrite is used, and the message is written into the message buffer.

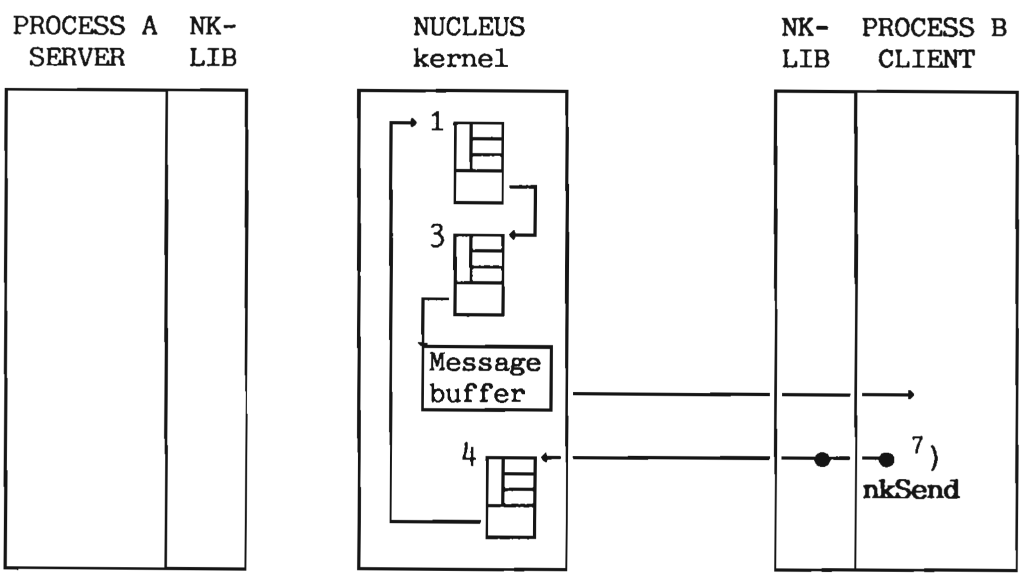



*Figure 31. Send a message*

[7]) Send the message to the port owned by process A. The message is appended at the end of port's message queue.
The sendreference is used to decide which port that is to receive the message.

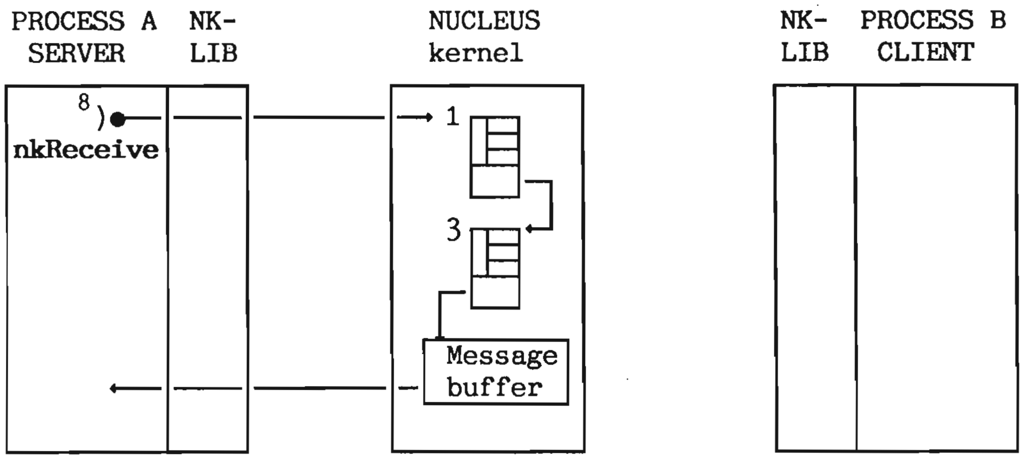



*Figure 32. Receive a message*

8) The first message in the queue is received. After the nkReceive call, the message is removed from the port's message queue. Process A becomes the owner of the message.

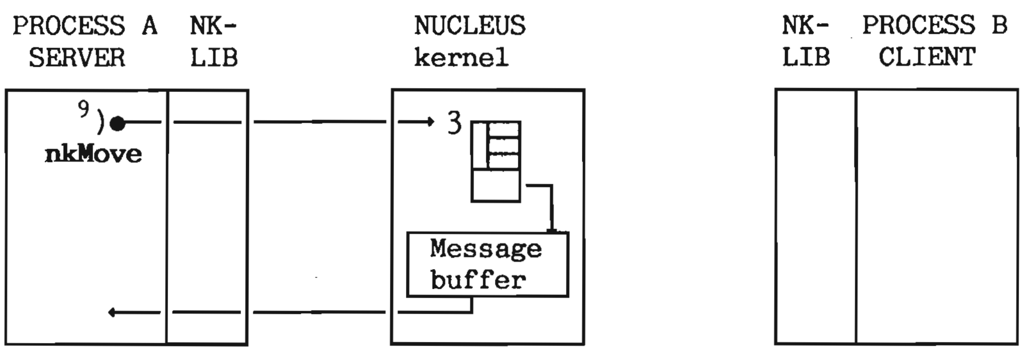



*Figure 33. Read a message*

9) The function nkfRead is used, and the message is read from the message buffer.

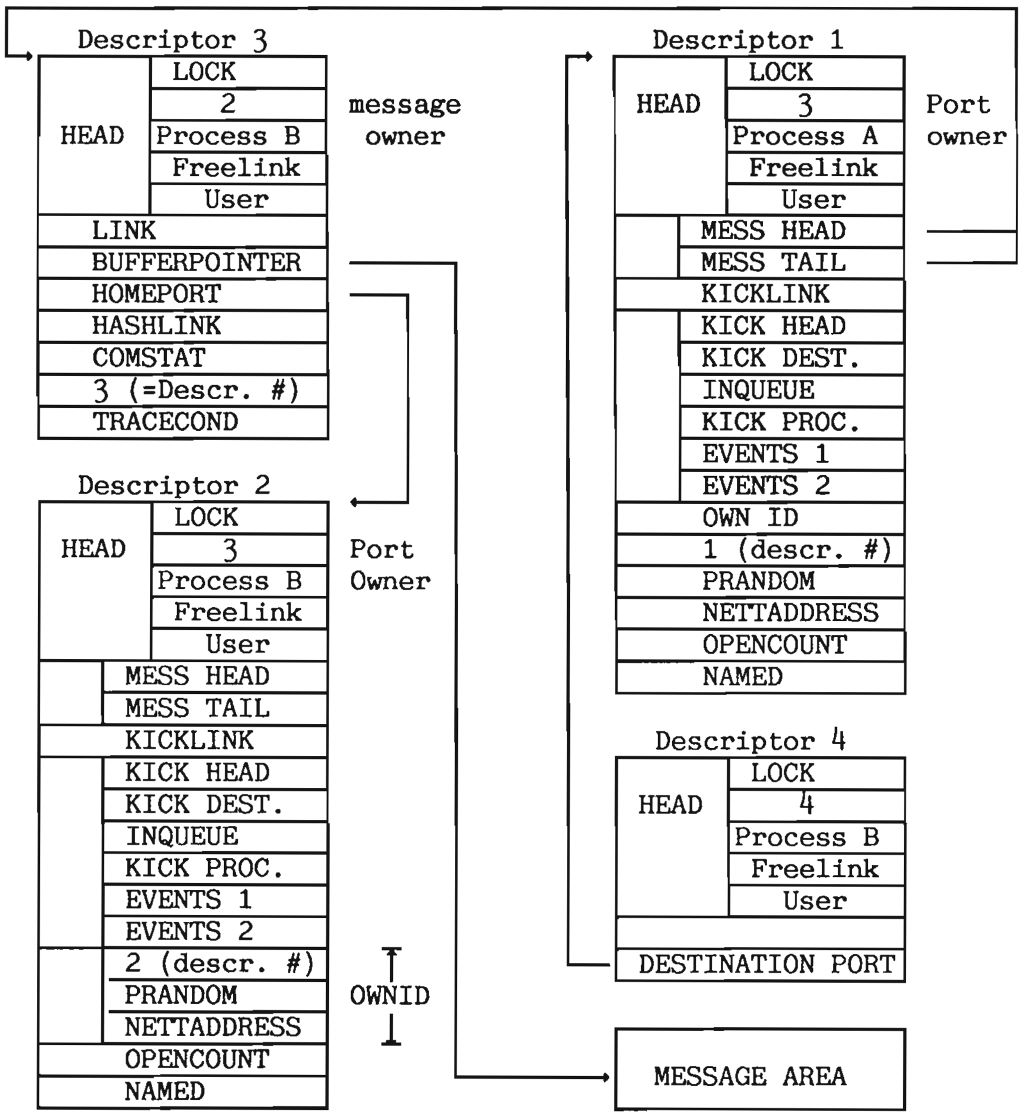
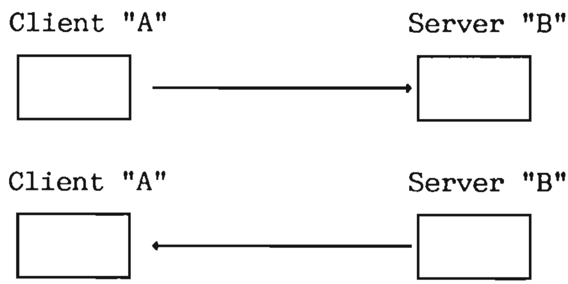



*Figure 34. Pointers in descriptor table*

# Chapter 8 PLANC Programming example

This chapter gives some examples of simple client/server cases. In real life, clients and servers will normally be in different processes. However, for simplicity all examples run as one single process here.

**Example 1:** This is a simple example where a client "A" sends a request to server "B" which responds.

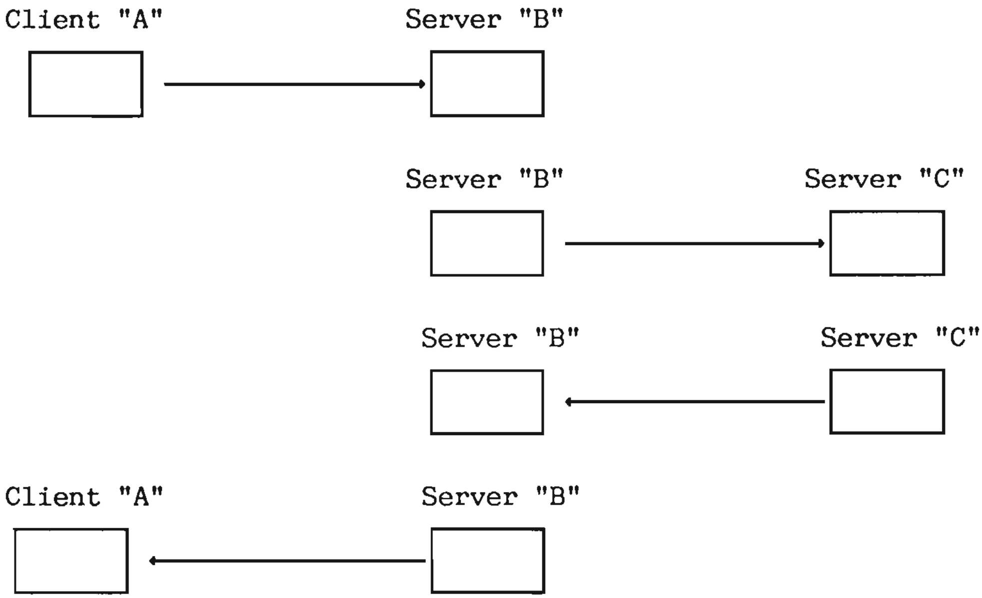



```
MODULE test

$INCLUDE (user-area)nk-library:impt

INTEGER ARRAY : stack (0:9999)

% server data:
INTEGER4:     S1RecPort % port number to receive message
INTEGER4:     S2RecPort % port number to receive message
INTEGER4:     S1Sendref % sendreferance  for serverA
INTEGER4:     S2Sendref % sendreferance  for serverB
INTEGER4:     smess % message number
INTEGER4:     sbmoved % number of bytes
INTEGER4:     smesslength % message length
BYTES:        quest(0:29) % received data

% user data:
INTEGER4:     URecPort % port number to receive message
INTEGER4:     umess % message number
INTEGER4:     USendref % sendreference
INTEGER4:     bmoved % number of bytes
INTEGER4:     umesslength % message length
BYTES:        response(0:29) % received data
```

```
%-----------------------------------------------------------------%
%-    C H S T A T                                                 %
%-----------------------------------------------------------------%
ROUTINE INTEGER4,VOID : chstat
  IF @><0 THEN @ ERRETURN ENDIF
ENDROUTINE

PROGRAM : testit
   INISTACK stack

   ON ROUTINEERROR DO
      Output (1,'a','$Routineerror in cliserv: ')
      Output (1,'o',ERRCODE)
   ENDON

   nkCrePort(0,2,S1RecPort)  chstat
   nkCreName(0,1,Addr 'serverA', S1Recport)  chstat

   % USER ESTABLISH CONNECTION
   nkCrePort(0,2,URecPort)  chstat
   nkCreMessage(0,50,URecPort,umess)  chstat
   nkOpenPort(0,Addr 'serverA',USendref)  chstat

   % USER REQUEST
   nkMove(1,umess,0,Addr 'ask serverA',bmoved)  chstat
   IF bmoved >< 11 THEN 1 ERRETURN ENDIF
   nkSend(0,0,USendref,umess)  chstat

   % SERVER READS REQUEST
   nkReceive(0,S1RecPort,smess,smesslength)  chstat
   IF smesslength >< 11 THEN 1 ERRETURN ENDIF
   nkMove(0,smess,0,Addr quest,sbmoved) chstat
   IF sbmoved >< 11 OR quest >< 'ask serverA' THEN 1 ERRETURN ENDIF

   % SERVER RESPONSE
   % SEND MESSAGE TO HOMEPORT OF MESSAGE
   nkMove(1,smess,0,Addr 'answer from serverB', sbmoved)  chstat
   IF sbmoved >< 19 THEN 1 ERRETURN ENDIF
   nkSend(0,0,0,smess)  chstat

   % USER GETS RESPONSE
   nkReceive(0,urecport,umess,umesslength)  chstat
   IF umesslength >< 19 THEN 1 ERRETURN ENDIF
   nkMove(0,umess,0,Addr response, bmoved)
   IF (bmoved >< 19) OR (response >< 'answer from serverB') THEN 1 ERRETURN
ENDIF

   % USER DISCONNECT
   nkClose(0,Usendref)  chstat
   nkClose(0,umess)     chstat
   nkClose(0,urecport)  chstat

   % SERVER DISCONNECT
   nkDelName(0,Addr 'testserv',S1Recport)  chstat
   nkClose(0,S1Recport)  chstat

   Output(1,'A','$ - CliServ session finished -')

ENDROUTINE
ENDMODULE
$EOF
```

**Example 2:**

1. Client "A" sends a request to server "B".
2. Server "B" sends the request to server "C".
3. Server "C" responds on the request from server "B".
4. Server "B" responds to client "A".
5. Server "C" respons to server "B" by means of last port set by server "B".




```
MODULE test

$INCLUDE (user-area)nk-library:impt

INTEGER ARRAY : stack (0:9999)

% server data:
INTEGER4:     S1RecPort % port number to receive message
INTEGER4:     S2RecPort % port number to receive message
INTEGER4:     S1UsrSendRef % sendreference for ServerA to User
INTEGER4:     S1Sendref % sendreference for ServerA
INTEGER4:     S2Sendref % sendreference for ServerB
INTEGER4:     smess % message number
INTEGER4:     sbmoved % number of bytes
INTEGER4:     smesslength % message length
BYTES:        quest(0:29) % received data

% user data:
INTEGER4:     URecPort % port number to receive message
INTEGER4:     umess % message number
INTEGER4:     USendref % sendreference
INTEGER4:     bmoved % number of bytes
INTEGER4:     umesslength % message length
BYTES:        response(0:29) % received data
```

```
%------------------------------------------------------------%
%-    C H S T A T                                            %
%------------------------------------------------------------%
ROUTINE INTEGER4,VOID : chstat
  IF @><0 THEN @ ERRETURN ENDIF
ENDROUTINE

PROGRAM : testit
   INISTACK stack

   ON ROUTINEERROR DO
      Output (1,'a','$Routineerror in cliserv: ')
      Output (1,'o',ERRCODE)
   ENDON


   % ServerA establish connection
   nkCrePort(0,2,S1RecPort)  chstat
   nkCreName(0,1,Addr 'ServerA',S1RecPort)  chstat

   % ServerB ESTABLISH CONNECTION
   nkCrePort(0,2,S2RecPort)  chstat
   nkCreName(0,1,Addr 'ServerB',S2RecPort)  chstat

   % USER ESTABLICH CONNECTION TO ServerA
   nkCrePort(0,2,URecPort)  chstat
   nkCreMessage(0,50,URecPort,umess)  chstat
   nkOpenPort(0,Addr 'ServerA',USendref)  chstat

   % USER REQUEST TO ServerA
   nkMove(1,UMess,0,Addr 'ask ServerA',bmoved)  chstat
   IF bmoved >< 11 THEN 1 ERRETURN ENDIF
   nkSend(0,Urecport,USendRef,UMess)  chstat

   % ServerA READ REQUEST
   nkReceive(0,S1RecPort,umess,SMessLength)  chstat
   nkOpenReturnPort(1,UMess,S1UsrSendRef)   chstat
   IF SMessLength >< 11 THEN 1 ERRETURN ENDIF
   nkMove(0,umess,0,Addr quest,sbmoved)  chstat
   IF sbmoved >< 11 OR quest >< 'ask ServerA' THEN 1 ERRETURN ENDIF

   % ServerA ESTABLICH CONNECTION TO ServerB
   nkOpenPort(0,Addr 'ServerB',S1Sendref)  chstat

   % ServerA REQUEST TO ServerB. ServerA SETS ITS OWN RECEIVE PORT
   % AS LAST SENDER.
   nkMove(1,umess,0,Addr 'ask ServerB',bmoved)  chstat
   IF SMessLength >< 11 THEN 1 ERRETURN ENDIF
   nkSend(0,S1Recport,S1SendRef,umess)  chstat

   % ServerB READ REQUEST FROM ServerA
   nkReceive(0,S2RecPort,umess,SMessLength)  chstat
   IF SMessLength >< 11 THEN 1 ERRETURN ENDIF
   nkMove(0,umess,0,Addr quest,sbmoved)  chstat
   IF sbmoved >< 11 OR quest >< 'ask ServerB' THEN 1 ERRETURN ENDIF

   % ServerB ESTABLISH CONNECTION TO ServerA
   % OPEN A SEND REFANSE TO PORT SET A LAST PORT IN MESSAGE.
   nkOpenReturnPort(1,UMess,S2SendRef)  chstat
```

```
   % ServerB RESPONSE ServerA
   nkMove(1,umess,0,Addr 'answer from ServerB',sbmoved)  chstat
   IF sbmoved >< 19 THEN 1 ERRETURN ENDIF
   nkSend(0,0,S2SendRef,umess)  chstat

   % ServerA READ REQUEST FROM ServerB
   nkReceive(0,S1RecPort,umess,SMessLength)  chstat
   IF SMessLength >< 19 THEN 1 ERRETURN ENDIF
   nkMove(0,umess,0,Addr quest,sbmoved)  chstat
   IF (sbmoved >< 19) OR (quest >< 'answer from ServerB') THEN 1 ERRETURN ENDIF

   % ServerA RESPONSE USER
   nkMove(1,umess,0,Addr'answer from ServerA',sbmoved)  chstat
   IF sbmoved >< 19 THEN 1 ERRETURN ENDIF
   nkSend(0,0,S1UsrSendRef,umess)  chstat

   % USER GETS RESPONSE FROM ServerA
   nkReceive(0,URecPort,umess,SMessLength)  chstat
   IF SMessLength >< 19 THEN 1 ERRETURN ENDIF
   nkMove(0,UMess,0,Addr quest,sbmoved)  chstat
   IF (sbmoved >< 19) OR (quest >< 'answer from ServerA') THEN 1 ERRETURN ENDIF

   % USER DISCONNECT
   nkClose(0,USendref)
   nkClose(0,Umess)
   nkClose(0,URecPort)

   % ServerA DISCONNECT
   nkClose(0,S1UsrSendref)
   nkClose(0,S1Sendref)
   nkDelName(0,Addr 'ServerA',S1RecPort)
   nkClose(0,S1RecPort)

   % ServerB DISCONNECT
   nkClose(0,S2Sendref)
   nkDelName(0,Addr 'ServerB',S2RecPort)
   nkClose(0,S2RecPort)

   Output(1,'A','$ - Example number 2 finished -')

ENDROUTINE
ENDMODULE
$EOF
```

**Example 3:** In this example, client "A" sends a request to server "B", and server "B" reads the message. Client "A" does not want a response from server "B". This is obtained by setting dump as home port when client "A" creates the message.


```
MODULE test

$INCLUDE (user-area)nk-library:impt

INTEGER ARRAY : stack (0:9999)

% server data:
INTEGER4:     S1RecPort % port number to receive message
INTEGER4:     S2RecPort % port number to receive message
INTEGER4:     S1Sendref % sendreferance  for ServerA
INTEGER4:     S2Sendref % sendreferance  for ServerB
INTEGER4:     smess % message number
INTEGER4:     sbmoved % number of bytes
INTEGER4:     smesslength % message length
BYTES:        quest(0:29) % received data

% user data:
INTEGER4:     URecPort % port number to receive message
INTEGER4:     umess % message number
INTEGER4:     USendref % sendreference
INTEGER4:     bmoved % number of bytes
INTEGER4:     umesslength % message length
BYTES:        response(0:29) % received data
```

```
%-------------------------------------------------------------------%
%-    C H S T A T                                                    %
%-------------------------------------------------------------------%
ROUTINE INTEGER4,VOID : chstat
  IF @><0 THEN @ ERRETURN ENDIF
ENDROUTINE

PROGRAM : testit
   INISTACK stack

   ON ROUTINEERROR DO
      Output (1,'a','$Routineerror in cliserv: ')
      Output (1,'o',ERRCODE)
   ENDON


   % ServerA ESTABLISH CONNECTION
   nkCrePort(0,2,S1RecPort)  chstat
   nkCreName(0,1,Addr 'ServerA',S1RecPort)  chstat

   % USER ESTABLICH CONNECTION TO ServerA. DUMP PORT IS SET
   % TO DEFAULT HOMEPORT IN NKCREMESSAGE.
   nkCrePort(0,2,URecPort)  chstat
   nkCreMessage(0,50,0,umess)  chstat
   nkOpenPort(0,Addr 'ServerA',USendref)  chstat

   % USER REQUEST TO ServerA
   nkMove(1,UMess,0,Addr 'ask ServerA',bmoved)  chstat
   IF bmoved >< 11 THEN 1 ERRETURN ENDIF
   nkSend(0,0,USendRef,UMess)  chstat

   % ServerA READ REQUEST
   nkReceive(0,S1RecPort,umess,SMessLength)  chstat
   IF SMessLength >< 11 THEN 1 ERRETURN ENDIF
   nkMove(0,umess,0,Addr quest,sbmoved)  chstat

   % USER DISCONNECT
   nkClose(0,USendref)
   nkClose(0,URecPort)

   % ServerA DISCONNECT. MESSAGE RECEIVED WILL BE DEALLOCATED.
   nkClose(0,Umess)
   nkDelName(0,Addr 'ServerA',S1RecPort)
   nkClose(0,S1RecPort)

   Output(1,'A','$ - Example number 3 finished -')

ENDROUTINE
ENDMODULE
$EOF
```

# Chapter 9 Error handling in NUCLEUS

## 9.1 NUCLEUS start up (system booting)

During start up of NUCLEUS in a DOMINO controller, NUCLEUS checks that:

- the correct version is installed, and that
- the controller address of NUCLEUS kernel is correct.

If a failure occurs during start up, an error message is sent to the Processor Manager, which writes an error message on the error device.

## 9.2 NUCLEUS fatal errors

When a fatal error occurs in NUCLEUS it is most likely that some memory conflict has occurred (NUCLEUS kernel area is overwritten by a DMA, system processes in the DOMINO controller etc..).

The error status identifying the cause of the error is sent to the Processor Manager server in ND-100 by means of an Octobus multibyte message and then written on the error device.

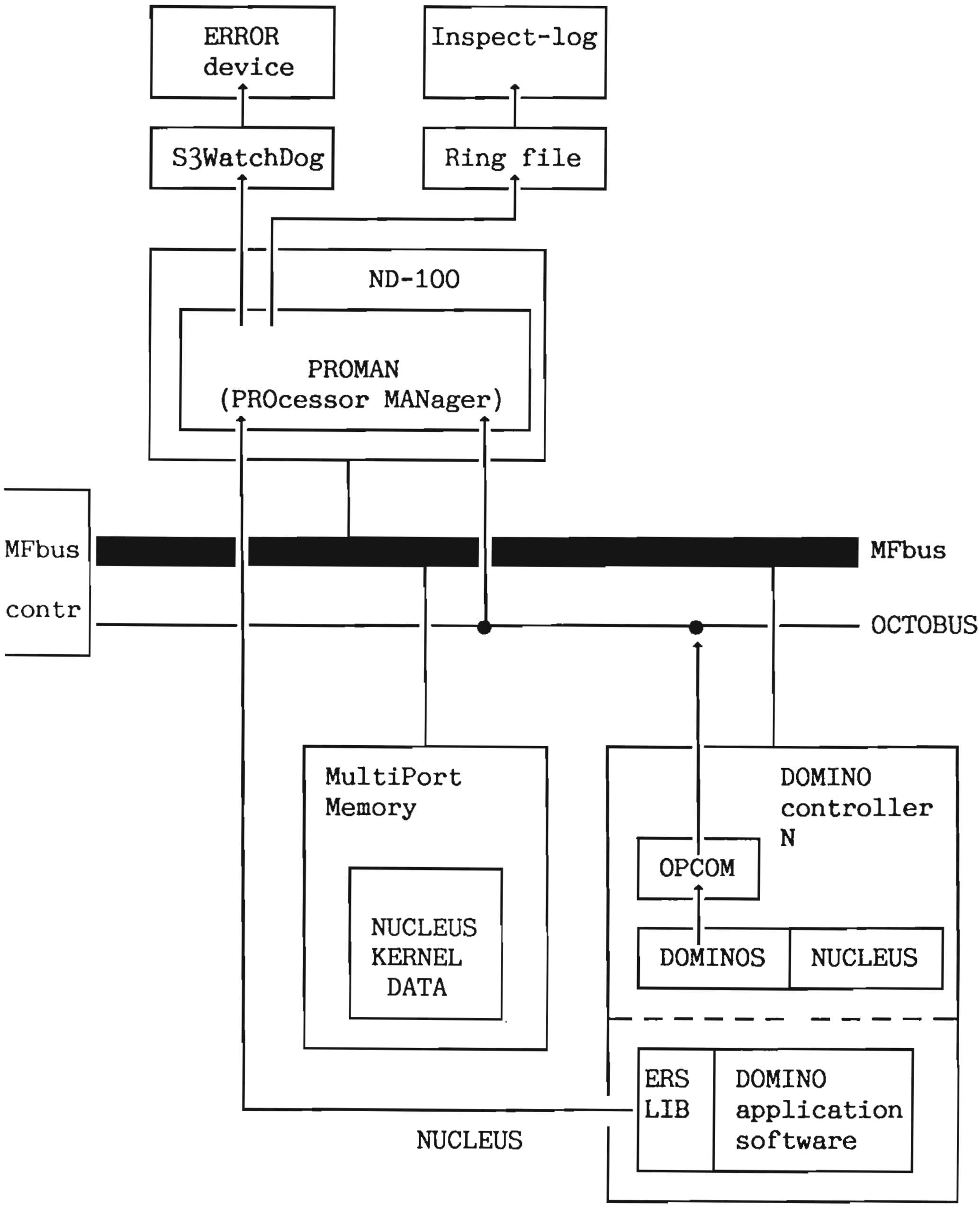



*Figure 35. Error in NUCLEUS*

## 9.3 NUCLEUS nonfatal errors

A nonfatal error will not corrupt the NUCLEUS kernel area. An octobus multibyte message is sent to PROMAN (Processor Manager), and then written on the error device.

## 9.4 Power failure handling

When a power fail occurs it is presumed that all CPUs are affected simultaneously. Multiple power sources and failures are not taken care of.

NUCLEUS may have set a Lock in the NUCLEUS kernel when a power failure occured. If not all CPUs are restarted at same time, it may cause a failure in NUCLEUS. NUCLEUS waits for a certain time to set a lock. If NUCLEUS is not able to set a lock, an error status will be returned. In the case of power failure this is very likely to occur.

Handling of power failure in NUCLEUS:

A global flag is set in NUCLEUS masterblock to indicate a powerfail at power-down. If a CPU is waiting for a lock, this flag is checked to see if the timeout has to be increased. When a CPU is recovered, this flag is reset and normal lock timeout is used.

The new lock routine in NUCLEUS is updated to give timeout. If power failures have occurred, the timeout limits are increased.

## 9.5 Verifications tests during start up

NUCLEUS checks if the NUCLEUS version number in each CPU is correct. In case of a version mismatch, an error message is sent to error device.

## 9.6 NUCLEUS verification program

The NUCLEUS verification program is delivered with the OS-kit, and runs as a background program in ND-100. The program is easy to use for debugging purposes. For each server, i.e. ND-100, ND-500 and DOMINO, separate programs (running as RT-programs) must be loaded. Please consult the PD-sheet. Log status from the servers are displayed on the screen, and saved on the log file NKS-LOGFILE:LOGS if any errors occur.

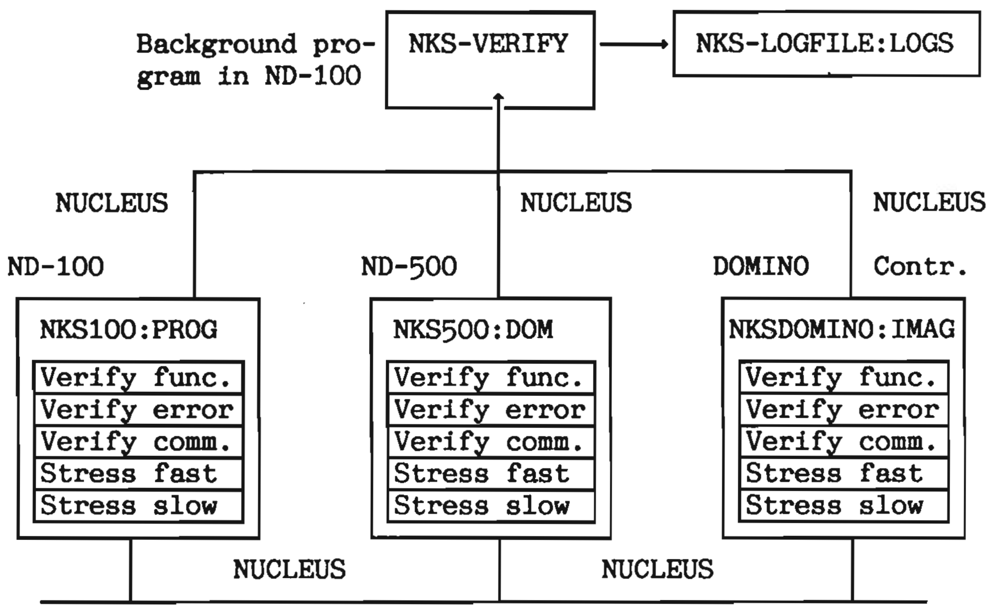



*Figure 36. NUCLEUS verification program*

To start the program give the command:

@(UTILITY)NKS-VERIFY↵

The screen picture shown below will now appear. An error message is displayed on the status line if you try to start logging from a server which is not loaded/started.

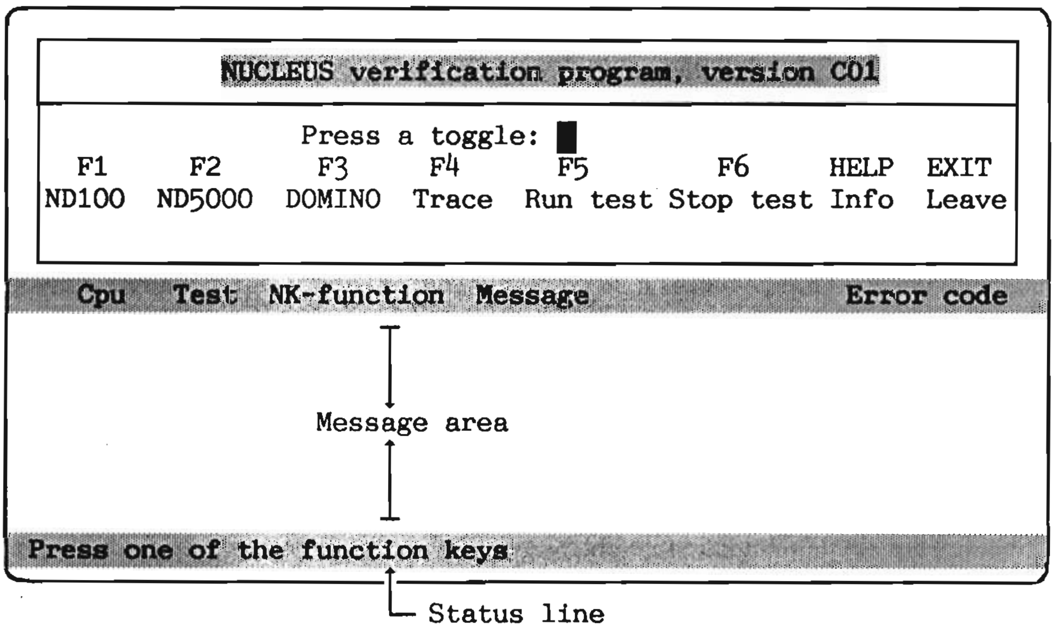



*Figure 37. NUCLEUS verification program - screen picture*

Messages are written in the message area when an error occurs, or when a test-module (see next page) is loaded/started/terminated.

**HELP facilities**

Press the HELP-key, and you will get information about:

- How to load,start and run tests,
- toggle status, and
- the NUCLEUS verification system.

**DOMINO station**

If you start verification in DOMINO (press the F3-key), the cursor will move, and you are asked to give the number of the DOMINO controller for which you want verification to be started.

> **NOTE !**
> DOMINO station is rebooted

**Test modules**

As the drawing on page 230 indicates, each server program consists of five modules:

1. Verify function.
2. Verify error handling.
3. Verify communication.
4. Stress the fast services, i.e. NkMove, NkSend and NkReceive.
5. Stress the slow services, i.e. NkCreMessage and NkCrePort

> **NOTE !**
> Stress modules are not implemented in C version

**Function keys**

| NOTIS terminal | non-NOTIS terminal |
|---|---|
| F1 | CTRL+D twice |
| F2 | CTRL+L |
| F3 | FUNC CTRL+U |
| F4 | FUNC + |
| F5 | FUNC Y |
| HELP | FUNC ? |
| EXIT | FUNC # |

**Simultanious verification**

Simultanious verification in ND-100, ND-500 and DOMINO is allowed. Verification can only be performed in one DOMINO at the time.

If you start verification in more then one server, you should not start tracing (F4) before you have inspected the log file. When you have found which server that produced the error message (the name of the server on the screen will blink), adopt the following course of action:

1. @(UTILITY)NKS-VERIFY↵ (start verification)

2. Press the toggle(Function-key) that corresponds to the server which produced the error message. If you start verification in DOMINO, you will also have to give the DOMINO station number.
3. Press the F4-key, in order to enable trace.
4. Press the F5-key, in order to run the test.

## 9.7 Debugging and tracing of NUCLEUS

NkCreMessage, NkSend, nkReceive and NkClose calls may be logged. Trace may be selected for one or more messages in NUCLEUS monitor. (See SET-TRACE on page 248 and LIST-TRACE on page 248).The trace element is put in a ring buffer in NUCLEUS kernel, and may be investigated by the NUCLEUS monitor.

# Chapter 10 NUCLEUS Monitor

**General**

The NUCLEUS Monitor is a tool for

- inspection of tables and queues in NUCLEUS kernel
- interactive use of NUCLEUS calls.

A typical use is to LIST ports or messages. More detailed information may be obtained by various DISPLAY commands. The data structures may be shown explicit by the LOOK-AT command. The monitor is also able to invoke NUCLEUS with the different functions like CREATE-PORT, CREATE-MESSAGE, SEND-MESSAGE...

The monitor has a HELP command that shows the possible commands and appropriate parameters. The LOOK-AT command has a HELP command as well.

## 10.1 Installation of NUCLEUS Monitor

The monitor is named **NK-MONITOR**, and recides on a floppy with directory DOMINO-KIT-C-5, and user-name FLOPPY-USER. Enter the floppy, and use Linkage-Loader to install the monitor.

There is an absolute correspondence between the NK-MONITOR and the current version of NUCLEUS. So this monitor will only work with the C version of NUCLEUS.

## 10.2 The command system

To start NUCLEUS Monitor, give the command:

**@ND-500 NK-MONITOR↲**

**Promt:** The NUCLEUS monitor prompts with **nkm:** whenever it is ready to accept a command. You may now use the commands on the high-level. If the commands you need are on the low-level(advanced mode), give the command:

**nkm:ADVANCED-MODE↲**

The monitor prompt changes to **nkm(adv):**

**Notation:** When describing the commands available in the NUCLEUS monitor, the following rules apply:

All parameter names are enclosed in <> brackets.

If a parameter that is asked for has a default value, the default value is enclosed between slashes //.

**NOTE !**

In commands with default values for descriptors ( i.e port, sendreference or message), the current descriptor is default If the default is not used, the given descriptor value also becomes the current descriptor.

The names of optional parameters, that are not asked if not given, are enclosed in square brackets. []

**Command entering** As in SINTRAN.

**Radix:** Numeric arguments may be given in octal, decimal or hexadecimal. The default radix is octal, but may be changed by the Main-format command. A trailing B (octal) or D (decimal) overrides the format.

## 10.3 NUCLEUS monitor commands

The description of the commands are divided into three parts:

- Commands common to high-level and low-level.
- Commands available on high-level only.
- Commands available on low-level only (advanced mode).

### 10.3.1 NUCLEUS monitor - common commands

#### Exit

High-level: Terminate execution of the NUCLEUS monitor.
Low-level : Return to high-level.

#### Main-format <format>

Define the main format for numbers displayed by the other commands. The format does not affect the numbers displayed by the Look-at command. See the Extra-format command.

Parameter: The wanted format. H, O and D is available. Only one may be used at the time.

H - Hexadecimal
O - Octal
D - Decimal

## Get-port-name <portnumber>

Displays all the port names defined for port number <parameter>.

Parameter: port number.

Format:

```
Port number:      4B   Name AAAA
```

## Help <command>

Displays available commands with their parameters on current level. Command names may be abbreviated as in SINTRAN.

## List-messages

Lists the messages with their message (descriptor) indices, owner ID and home ports.

Format:

```
Message:         Owner:          Homeport:
     7B      100275031B                6B
    11B      100275041B               10B
```

## List-names

Displays all the port names defined in the name server, with their corresponding port number, random number and netaddress.

Format:

| Port | Random : | NetAdr | Port Name |
|---|---|---|---|
| 11B | 7B : | 40B | AAAA |
| 10B | 6B : | 40B | NKMTDRIVER |
| 7B | 5B : | 40B | NKMTSERVER |
| 6B | 4B : | 40B | PMAersGateWay |
| 5B | 3B : | 40B | PMAservicePort |
| 4B | 2B : | 40B | PMAhomePort |
| 3B | 2B : | 40B | serviceport |

One port may have more than one name, but two ports cannot share a name. If a server terminates, the port names will still be present in the nameserver unless they are deleted by the termination process, or by NUCLEUS.

It may look as if a port has more than one name, especially if a process fail to terminate properly. In most cases, this is not true, as the random number part of the port number is different.

## List-ports

Lists the ports with their port (descriptor)number, owner ID, number of messages and number of home messages.

Format:

| Port number: | Owner: | Messages: | Home messages |
|---|---|---|---|
| 1B | 100062542B | 0B | 0B |
| 2B | 100062570B | 0B | 0B |

### Verify

This command performes a consistency check of the data structure. Inconsistencies are reported.

## 10.3.2 NUCLEUS Monitor - high-level commands

The commands described in this section, are available on this level only. Commands described in the previous section are also available.

### Advanced-mode

Gives the user acccess to the low-level commands. See the next section, starting on page 244.

### Close <descriptor>

Close a port,sendreference or message defined by <parameter>.

Parameter: Descriptor number(index).

No default value.

## Create-message <size><homeport>

Creates a message of size <parameter 1> with homeport <parameter 2>. The message number of the message is returned. The message becomes the current message.

Parameter 1: size of message
Parameter 2: homeport for the given message.

## Create-name <name><port>

Creates port name <parameter 1> on port <parameter 2>.

Parameter 1: Port name.
Parameter 2: Port number. /current port/

**NOTE !**
In commands with default values for port number, the current port number is always the default port number. If the default is not used, the given port number becomes the current port number. This also applies for message and sendreference

## Create-port

Creates a port. A port number is returned, and the port becomes the current port.

## Fill-data-buffer <string>

Fill the buffer with the string <parameter>. The string, given as parameter, is moved to the buffer area in the monitor. The buffer area may written into the NUCLEUS area, and sent to a destination port by the Write-message and Send-message command.

Parameter: any string.

Note: The command is meaningless in the B00 version of the monitor, as the Write-message command also fills data into the buffer.

## Open-port <port name>

Open the port with name <portname>.
A sendreference number is returned, and can be used to send to <portname>.
The sendreference number returned, becomes current sendreference.

## Print-data-buffer

Displays the content of the send/receive buffer in the monitor. The content is displayed in both octal and ascii format.

**Example:**

```
Data buffer content:
   0B :   0B   0B  74B 375B 102B 102B 102B 102B      (..<.BBBB)
  10B : 102B 102B 102B 102B 102B 102B 102B 102B      (BBBBBBBB)
  20B : 102B 102B 102B 102B 102B 102B 102B 102B      (BBBBBBBB)
```

## Receive-message <port no.>

Receive message from <portno.>. Parameter: port number /current port/.

## Read-message <message><displacement>

Read message from <messagenumber> to data buffer, start from position <displacement>

Parameter 1: message number /current message/.
Parameter 2: displacement in message buffer /0/.

The received dat will be displayed, and may be redisplayed with the PRINT-BUFFER-command.

## Send-message <port no.><message no.>

Send <message number> to <port number>

Parameter 1: port number to send message to /?/.
Parameter 2: message number to send /curr message/.

## Write-message <message no><displacement><text>

Write <messagenumber> to data buffer, start in position <displacement>

Parameter 1: message number /current message/.
Parameter 2: displacement in message buffer /0/
Parameter 3: any string of text.

## 10.3.3 NUCLEUS Monitor - low-level commands

The commands described in this section, are available on low-level only. Commands described in the section "NUCLEUS Monitor - common commands" are also available. See page 237.

### Connect-file <file name>

Parameter: File-name. Default file type is **:DUMP**.

The connect-file command is intended to be used to investigate a dump of NUCLEUS kernel. Most low-level commands can be used (display/list commands).

The dump file can be made by means of the DUMP-KERNEL command. You may also use the stand-alone program **MEMTOF**, and dump the memory to a diskette.

### Display-descriptor <descriptor index>

Descriptor may be port, sendreference or message.
Parameter: descriptor index(number).

### Display-kicklist <OCTOBUS station no.>

Display the kicklist for Octobus station <parameter 1>.

Parameter: Octobus station number.

The kicklist is a list of receive ports. Processes owning ports in the kicklist, will be activated.

Example:

```
Port                    4b
Port                    7b
```

## Display-masterblock

Displays the masterblock for NUCLEUS. The index limits for the descriptor array, hash array, kick table and net table is displayed. Of these, only the descriptor array and kick table have meaning, as NUCLEUS communication is not implemented.

```
nkm:ADVANCED-MODE↲
nkm(adv):DISPLAY-MASTERBLOCK↲

   Masterblock address:        20000004000B
   Version:                            103B
   Protocoll:                           2B
   Descriptor array:           20000004244B      ( 0B :   777B)
   Hash array:                 20000104244B      ( 0B :   377B)
   Hash mask                           377B
   Kick table:                 20000106244B      ( 0B :    77B)
   Net table:                  20000107644B      ( 0B :    11B)
   Trace buffer:               20000107764B      ( 0B :   377B)
   Trace pointer:                       0B
   Quota table:                20000117764B       (0B :    61B)
   Quota hash array:           20000122244B       (0B :   377B)
   Quota hash mask:                    377B
   Quota free link header:     20000122214B
   Free link header            20000006044B
   Buffer area start:          20000124244B
   Buffer area end:            20002624240B
   Local net address:                  40B
   Power fail 1:                        0B
   Power fail 2:                        0B
   Current random number:             472B
   Trace condition:                     0B
   General area lock:                   0B
   Hash lock:                           0B
   Trace lock:                          0B
   Allowed public descript.:          300B
   Allowed public buffer:         1240000B
   Used public descriptors:             0B
   Used public buffer:                  0B
   Message free list:                   0B
   Number of free messages:             0B
nkm(adv):
```

## Display-messages <message>

Lists data related to message <message>

Parameter: message number /current message/.

## Display-port <portnumber>

Lists data related to port <portnumber>.
Parameter: port number /current port/.

**Example:**

```
nkm(Adv): DISPLAY-PORT↵
Portnumber: 7B↵

Owner                   :    100411013B
Octobus station         :            1B
Process to be kicked    :    100411013B
Event set               :            1B
Open count              :            1B
Decriptor address       :  20000445144B
```

## Dump-kernel <file name>

Dump the NUCLEUS kernel to <file name>. Parameter: File name. See also the related command CONNECT-FILE on page 244.

## Extra-formats <format>

Define extra-formats for the Look-at command. Parameter: extra format string. HDOA or any combination of them may be used.

H - Hexadecimal
D - Decimal
O - Octal
A - Ascii

## Force-display <descriptor index><type>

Display the <descriptor index>, **as if** it where of the type <type>. Parameter 1: Descriptor index (number). Parameter 2: Descriptor type.

Legal values for descriptor type:

0 = unused
1 = used
2 = message
3 = port
4 = sendreference

## Get-Nucleus-memory

This command is for special use, intended for internal debugging only.

## List-trace <number>

The last <number> trace elements are listed.

```
Parameter : <number>=0 => List all trace elements.
            <number>=i => List the i (i=1,2,..,n)
                          last trace elements.
```

## List-quota

Lists all users in the quota table.

## Look-at

This command is similar to the look-at-data command in the symbolic debugger.

## Set-trace <message><on/off>

Set trace for <message> number <off/on>.

```
Parameter 1: Number of message to be traced.
             <message>=0 => All new messages are
             traced.
Parameter 2: 0=Off.
             1=On.
```

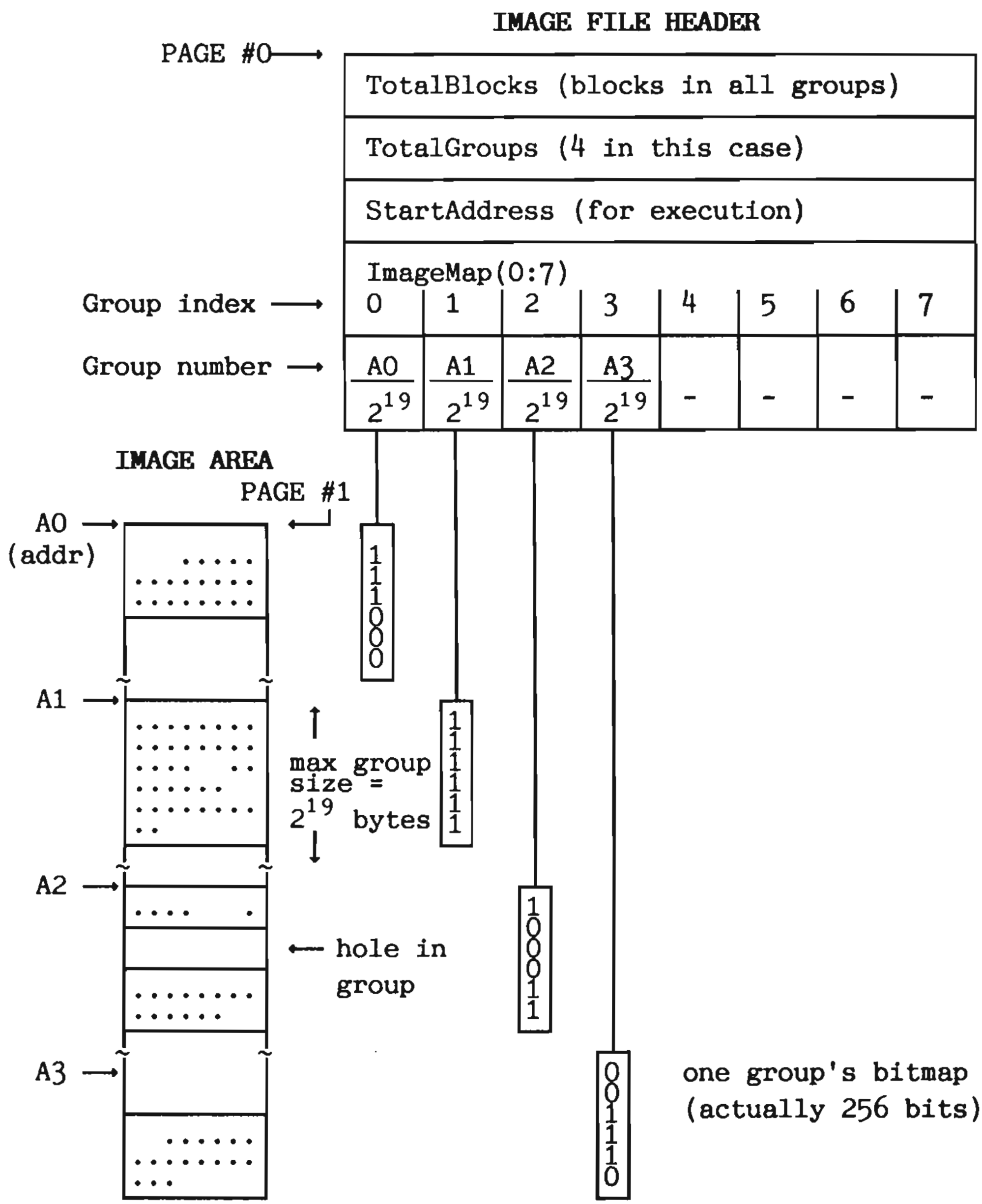
# Appendix A Image files

**Groups blocks and bytes**

An image is divided into groups, each of 256 blocks. Each group has a bitmap showing the blocks used. A block has 2048 bytes. An image may have eight groups. A group describes a continuous memory area of 512 K bytes. Each group may have a group number from 0 to 255, giving a potential address span of 128 M bytes.

An image may be scattered in the full address space. But, as only eight groups are allowed, all blocks in the image must be within the eight, even if not all blocks are used in all groups. The place address in DOMINO memory for a block is derived directly from the group and block number:

$$\text{PLACE ADDRESS} = \text{GROUPNUMBER}*2^{19} + \text{BLOCKNUMBER}*2^{11}$$




..... = program code/initialized data
The group numbers need not be in increasing order as in this example.

*Figure 38. Image file header versus image area*

# Appendix B   DOMINO selftests

**Test numbers**

All tests has a test number which must be reported at the start of the test, and when errors are found. The following reservation of test numbers is done.

| Test no. | Test type. |
|---|---|
| 1 - 2F | Preboot tests. |
| 30 - 7F | Standard postboot tests |
| 80 - EF | Device dependant tests |
| F0 - FF | Reserved |

> NOTE!
> All device dependant tests must start with test number 80 (hex)

**Selftest reporting to test connector**

All selftests reports to the test connector (Address: FF8104) when starting. This reporting is done to make it easy to trace the selftests on a logic analyzer or tracer. The selftest report to the test connector consists of two parts. The first byte is the test number and the second byte is an error number. A zero in the 'error byte' means that this is the start of the test (no error).

**Example**

The following would be a typical display on a tracer storing all cycles to the test connector.

| Address | Data | Meaning |
|---|---|---|
| FF8104 | 0100 | Start of selftest no. 1 |
| FF8104 | 0200 | Start of selftest no. 2 |
| FF8104 | 0300 | Start of selftest no. 3 |
| FF8104 | 0301 | Error no. 1 found in selftest no 3 |

When an error is found in a selftest, the test and error number is written to the test connector, and an error message is written to the current path (asyl or OCTOBUS).

The error message is on the following form:

SELFTEST ERROR NO : <test and error no.> <Text explaining what kind of error.>

**Example** SELFTEST ERROR NO : 101
Wrong checksum in DOMINO EPROM

**How to use a TDF** All selftests (except preboot) follows the defined layout using TDF's. A short description of all the elements in the TDF record is given below.

```
TYPE tdf = RECORD
   INTEGER4           : TdfKey               % Must be OkKey
   BYTES              : Name(0 : 9)
   tdf POINTER        : NextTdf
   INTEGER4           : RunsToGo, RepFreq, MaxNbErr
   TestRtn POINTER    : TestPtr
   RespRtn POINTER    : RespPtr
   Data POINTER       : InParams
   Data POINTER       : OutParams
   INTEGER4           : TxuErrCode           % 0=Ok, >< 0 Error
   INTEGER4           : DoneRuns, NbErrors
   ENUMERATION(TstIdle, TstRunning, TstAborted, TstErrReport, &
               TstMsgReport) &
                      : State
   INTEGER2           : Spare1
ENDRECORD
```

**TdfKey**
This should always be OkKey, where OkKey is defined as:
CONSTANT OkKey = 12421043040B % Ascii 'TDF '

**Name(0:9)**
This is the name of your test. (Choose a good one)
Ex. 'PROT-TEST '

**NextTdf**
Pointer to next TDF. This will usually be NIL for selftests.

**RunsToGo**
Number of times the test should be run. This should always be 1 for selftests.

**RepFreq**
Report frequenzy. Should always be 0 for selftest, because only errors should be reported.

**MaxNbErr**
Maximum number of errors allowed before test is terminated. Will usually be 1 for selftests.

**TestPtr**
Pointer to your testroutine.
Ex. ADDR ProtectTest
ProtectTest is the name of the routine containing your test.

**RespPtr**
Pointer to a response routine. This routine will be called when you are doing a TRAP£6 or TRAP£7. The routine should contain all error output from your test.
Ex. ADDR ProtectTstResp
ProtectTstResp is the name of a routine that outputs error messages from your test.

**InParams**
Pointer to a record containing parameters from NDITS to be used by the test. Will very often be NIL for selftests.

**OutParams**
Pointer to a record containing parameters from the test to NDITS. Will very often be NIL for selftests.

**TxuErrCode**
TEXU error code. Must be set to 0 for selftests.

**DoneRuns**
Number of times the test is actually run.
Must be set to 0 for selftests.

**NbErrors**
Number of errors found so far for this test.
Must be set to 0 for selftests.

**State**
State will be one of the following:
TstIdle, TstRunning, TstAborted, TstErrReport, TstMsgReport
Must be set to TstIdle for selftests.

Following is an example of a TDF array used by the postboot tests.

```
tdf ARRAY                : SelfTests(ProtTest : 2):=(&
(OkKey, 'PROT-TEST ', NIL, 1, 0, 1, Addr ProtectTest,
Addr ProtectTstResp, ProtTstIn, ProtTstOut, 0, 0, 0, TstIdle, 0),
(OkKey, 'Mfp-Test  ', NIL, 1, 0, 1, Addr MfpTest,
Addr MfpTstResp, NIL, MfpTstOut, 0, 0, 0, TstIdle, 0))
```

**Exception handling in selftests**

Unexpected exceptions in selftests are reported with the following error numbers. The error numbers are reported to the test connector, and a message is written to asyl/OCTOBUS.

| | |
|---|---|
| FCH | Parity error occured. |
| xxFDH | Exception occured in test xx |
| xxFEH | Int. 7 occured in test xx |
| xxFFH | Bus error occured in test xx |

**Preboot tests**

| Test Name | Test No/ Err. No | Description |
|---|---|---|
| Promtest | 100H | Start of prom checksum test. |
| -- | 101 | Wrong checksum in DOMINO prom.<br>- Something wrong with the prom?<br>- Collision with other devices on the data bus?<br>- Address lines connected together? |
| -- | 102 | Wrong checksum in the device prom.<br>- Something wrong with the prom? |
| MCR test | 200H | Start of Master Control Register test. |
| -- | 201 | MCR not zero after reset.<br>- Bad register package?<br>- Problems with data bus?<br>- Problems with addressdecoding of MCR? |
| -- | 202 | MCR readback error. Read data not equal to written data.<br>- Bad register package?<br>- Problems with write strobe to MCR? |
| Buserrortest | 300H | Start of buserror test. |
| -- | 301 | DSACK instead of BERR.<br>A bus error was forced, but instead the cycle was terminated with a DSACK.<br>- Problem in address decoding? |
| -- | 302 | Local timeout bit not set<br>- Problems with the BerrInt7 register? (INT7 PAL)<br>- Problems with the address decoding of the same register? |

| Test Name | Test No/ Err. No | Description |
|---|---|---|
| WarmCold | 400H | Start checking for warm or cold start. |
| AddrInAddr | 500H | Start of memory test. |
| -- | 501 | Error found in RAM<br>- Problems with timing against DRAM? Missing RAS, CAS RW etc.<br>- Problems with data or address bus?<br>- Problems with one of the memory packages? |
| Sizetest | 600H | Start of byte selection test.<br>All combinations of byte, word, long, read and write is tested.<br>- Problems with the RW pal that generates write strobes to the DRAM?<br>- Problems with timing against DRAM? Missing RAS, CAS RW etc. |
| -- | 601 | Write Byte, Read Byte err. |
| -- | 602 | Write Byte, Read Word err. |
| -- | 603 | Write Byte, Read Long err. |
| -- | 604 | Write Word, Read Byte err. |
| -- | 605 | Write Word, Read Word err. |
| -- | 606 | Write Word, Read Long err. |
| -- | 607 | Write Long, Read Byte err. |
| -- | 608 | Write Long, Read Word err. |
| -- | 609 | Write Long, Read Long err. |
| ParityErr | 700H | Start of parity test. |
| -- | 701 | Unexpected Parity Error<br>A parity error occured immediately after enabling the parity system.<br>- Problems somewhere in the parity network. |
| ParityNet | 800H | Start of parity check for each byte. |
| -- | 804 | No interrupt from byte 0<br>- Problems in parity network for byte 0? |
| -- | 805 | No interrupt from byte 1<br>- Problems in parity network for byte 1? |
| -- | 806 | No interrupt from byte 2<br>- Problems in parity network for byte 2? |
| -- | 807 | No interrupt from byte 3<br>- Problems in parity network for byte 3? |

| Test Name | Test No/ Err. No | Description |
|---|---|---|
| ParityNet | 808 | Parity error bit not set 0<br>- Problems with the BERRINT7 register. |
| -- | 809 | Parity error bit not set 1<br>- Problems with the BERRINT7 register. |
| -- | 80A | Parity error bit not set 2<br>- Problems with the BERRINT7 register. |
| -- | 80B | Parity error bit not set 3<br>- Problems with the BERRINT7 register. |
| ParityAddr | 900H | start |
| -- | 901 | - Error in parity RAM<br>or in address bits? |
| ParityData | A00H | start |
| -- | A01 | - Error in parity gen/check |
| Booting | B00H | Boot and switch RAM mode? |
| ParitySwitch | C00H | start |
| -- | C01 | No switch no interrupt |
| -- | C02 | No switch interrupt<br>- Problems in the MCR clear logic? |
| -- | C03 | Switch no interrupt<br>- Problems in the interrupt system? |
| 4-8 MB memtest | D00 | start |
| | D01 | Error found in RAM |
| Uart/TimD | E00 | start |
| -- | E01 | TimerC/D CtrlReg. ReadBackErr |
| -- | E02 | TimerD DataReg. ReadBackError |
| -- | E03 | Usart CtrlReg. ReadBackError |
| -- | E04 | Rx Status Reg. ReadBackError |
| -- | E05 | Tx Status Reg. ReadBackError |
| -- | E06 | Usart Data Reg. ReadBackError |
| -- | E07 | Rx StatReg does't stabelize |
| -- | E08 | Tx StatReg does't stabelize |
| -- | E09 | Usart Data Reg timeout |
| -- | E0A | Illegal value in rsr |
| -- | E0B | Illegal value in tsr |
| -- | E0C | Too many/few Rx interrupts |
| -- | E0D | Too many/few Tx interrupts |
| PrebootOK | 2F00H | Preboot test OK |

**Postboot tests**

```
Prot test       3000H  Start of protect test.
                30yy   The error code yy has the following
                       meaning from the protect test.

                       bit no. in yy
                       76543210
                       |||||||└── Protect mode bit S1
                       ||||||└─── Protect mode bit S2
                       |||||└──── Protect mode bit U1
                       ||||└───── Protect mode bit U2
                       ||||            See DOMINO HW desc. for more de-
                       ||||            tails about the prot. mode bits.
                       |||└────── FC0 for the cycle that failed
                       ||└─────── FC1 for the cycle that failed
                       |└──────── FC2 for the cycle that failed
                       |               See MC68020 User's manual for
                       |               more details about the
                       |               function codes.
                       └───────── RW  for the cycle that failed
                                       0=Write, 1=Read

MFP test        3100H  Start of MFP (MC68901) test.
                3101   Interrupts pending before they are enabled.
                       - Error in the MFP?
                3102   No interrupt pending after they are enabled.
                       - Error in the MFP?
                3103   Processor was not interrupted.
                       - Error in the MFP?
                       - Error in the interrupt system?
                3104   Timer error.
                       - Error in the MFP?
                3105   Too many interrupts.
                       - Error in the MFP?
```

Continue on next page...

| | | |
|---|---|---|
| EEPROM test | 3200 | Start of EEPROM check. |
| | 3201 | Not valid EEPROM testpattern.<br>- No EEPROM?<br>- EEPROM not initialized?<br>- Problems in reading from EEPROM? |
| | 3202 | Not valid EEPROM version.<br>- EEPROM not initialized correct?<br>- Problems in reading from EEPROM? |
| | 3203 | Write access to write protected area.<br>- Problem in decoding of write strobe to EEPROM? |
| | 3204 | Timeout. Busy signal from EEPROM constantly active.<br>- Problems with EEPROM? |
| Counter test | 3300 | Start of 32 bit counter test. |
| | 3301 | 16 bit counter is not running.<br>- Error in the counter? |
| | 3302 | 32 bit counter is not running.<br>- Error in the counter?<br>- No master selected on OCTOBUS? |
| | 3303 | Protect trap when read from user mode. |
| BADAP test | 3400 | Start of BADAP register test. |
| | 3401 | Readback error from BADAP register.<br>- Error in BADAP?<br>- Error in data path to BADAP? |

# Appendix C Error and status codes

## PROMAN (Processor Manager) error codes

| Octal val | Meaning | Type |
|---|---|---|
| 1050B | Processor Manager - PROMAN | |
| 105000B | Too large configuration | ERROR |
| 105001B | Program error, empty time queue | ERROR |
| 105002B | Unrecognised event ignored | WARN. |
| 105003B | Message from unrecognised Octobus station ignored | WARN. |
| 105004B | Image file is empty, booting aborted<br>Controller at station..: I20 | ERROR |
| 105005B | Unrecognised event in Domino boot-session ignored<br>Controller at station..: I20 | WARN. |
| 105006B | Program error, invalid dummy-session state | ERROR |
| 105007B | Program error,invalid DOMINO-session state | ERROR |
| 105007B | Program error,invalid DOMINO-session state<br>Controller at station..: I20 | ERROR |
| 105010B | Program error, invalid ERS-session state | ERROR |
| 105011B | Program error, invalid Service-session state | ERROR |
| 105012B | Unrecognised dummy-session event ignored | WARN. |
| 105013B | Unrecognised Domino-session event ignored<br>Controller at station..: I20 | WARN. |
| 105014B | Unrecognised ERS-session event ignored | WARN. |
| 105015B | Unrecognised Service-session event ignored | WARN. |
| 105016B | Error in Nucleus initialisation | FATAL |
| 105017B | Error in Octobus initialisation | FATAL |
| 105020B | Error in Nucleus receive | WARN. |
| 105021B | Error in Octobus receive | WARN. |
| 105022B | Error in Nucleus transmit | WARN. |
| 105023B | Error in Octobus transmit | WARN. |

Continue on next page...

**PROMAN (Processor Manager) error codes**

| Octal val | Meaning | Type |
|---|---|---|
| 105024B | Controller does not respond, echo-test failed<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105025B | Unable to get identity from controller<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105026B | Unable to stop controller<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105027B | Unable to set mailbox for controller<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105030B | Unable to download block to controller<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105031B | Unable to set start address in controller<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105032B | Unable to start program in controller<br>Controller at station..: I20<br>Opcom NAK-error code("0" means timeout)I20 | ERROR |
| 105033B | OPCOM selftest failed<br>Controller at station..: I20 | ERROR |
| 105034B | Invalid service request command | WARN. |
| 105035B | No Domino controllers found in system | WARN. |
| 105036B | Unable to log events to ring buffer file | WARN. |
| 105037B | This ND-100 is not master in system | FATAL |
| 105040B | Error when opening image file<br>Controller at station........: I20<br>Image file name..............: 64A | ERROR |

Continue on next page....

| Octal val | Meaning | Type |
|---|---|---|
| 105041B | Server started<br>Version: 64A | INFO |
| 105042B | DOMINO controller booting started<br>Controller at station........: I20<br>Crate ID (MFB contr station).: I2D<br>MFB Slot....................: I2D<br>Image file name.............: 64A | INFO |
| 105043B | DOMINO controller rebooting started<br>Controller at station........: I20<br>Crate ID (MFB contr station).: I2D<br>MFB Slot....................: I2D<br>Image file name.............: 64A | INFO |
| 105044B | DOMINO controller started<br>Controller at station........: I20<br>Crate ID (MFB contr station).: I2D<br>MFB Slot....................: I2D<br>Image file name.............: 64A | INFO |
| 105045B | DOMINO controller selftest status<br>Controller at station........: I20<br>Crate ID (MFB contr station).: I2D<br>MFB Slot....................: I2D<br>CPU type....................: I4D<br>Standard part version..: 34A<br>- Selftests failed.....: 64A<br>Device part version....: 34A<br>- Selftests failed.....: 64A | INFO |
| 105046B | Server stopped | INFO |
| 105047B | Domino controllers restarted after powerfail<br>Number restarted.......: I2D<br>Number rebooted........: I2D | INFO |
| 105050B | Too small ERS-buffer fil, must at least be two pages long | WARN. |
| 105051B | Domino controller has been terminated on request<br>Controller at station..: I20 | INFO |

*Table 8. PROMAN (Processor Manager) error codes*

**DOMINOS error codes**

| Constant | Octal value |
|---|---|
| PITermination | 6000B |
| PIILCAL | 6001B |
| PIRANGE | 6002B |
| PICONTX | 6003B |
| PISupModeCall | 6004B |
| PIintErr | 6005B |
| PIDomFatal | 6006B |
| PIUserFatal | 6007B |
| PINOEXIST | 6011B |
| PIEXIST | 6012B |
| PIILPRI | 6013B |
| PIILSTATE | 6014B |
| PINOPROS | 6015B |
| PINOFREE | 6016B |
| PIEVNOEX | 6021B |
| PIILVEC | 6022B |
| PINOBUF | 6041B |
| PIINCONSIST | 6042B |
| PIILADDR | 6043B |
| PINoRout | 6051B |

**DOMINOS, DOMINO Operating System errors**

| Octal | Meaning | TYPE |
|---|---|---|
| 60B | Domino Operating System | |
| 6000B | Application in DOMINO controller terminated<br>Octobus station.............. I20<br>Crate Id (MFB contr. station) I1D<br>MFB slot..................... I1D | INFO |
| 6001B | Requested service not implemented | ERROR |
| 6002B | Parameter value to service out of range | ERROR |
| 6003B | Request comes in wrong context | ERROR |
| 6004B | Service called in supervisor mode | FATAL |
| 6005B | Dominos program error, please contact ND-service | ERROR |

Continue on next page...

| Octal | Meaning | TYPE |
|---|---|---|
| 6006B | Unable to start Dominos<br>Octobus station.............. I20<br>Crate Id (MFB contr. station) I1D<br>MFB slot..................... I1D<br>Exception number............. I2unused<br>Address where occurred....... I4H<br>Dominos error................ SEC | FATAL |
| 6007B | Error reported by PIRFatal<br>Octobus station.............. I20<br>Crate Id (MFB contr. station) I1D<br>MFB slot..................... I1D<br>Exception number............. I2unused<br>Address where occurred....... I4H<br>Application error............ SEC | FATAL |
| | Process management | |
| 6011B | Process does not exist | ERROR |
| 6012B | Process already exists | ERROR |
| 6013B | Invalid priority in PIRCREATE or PIRMODIFY | ERROR |
| 6014B | Requested operation impossible in this process state | ERROR |
| 6015B | Invalid process name | ERROR |
| 6016B | No free entry for new process | ERROR |
| | Event system/timing/miscellaneous | |
| 6021B | Event not found | ERROR |
| 6022B | Invalid vector address, (outside 2..255 or reserved) | ERROR |
| | Buffer manager | |
| 6041B | Buffer space exeeded | ERROR |
| 6042B | Inconsistency in Buffer data structure | ERROR |
| 6043B | Invalid Buffer address | ERROR |
| | Powerfail/power return handling | |
| 6051B | Power fail/Power return handler not found | ERROR |

*Table 9. DOMINOS, DOMINO Operating System errors*

**DOMINO Services (HW-LIB/OPCOM) error codes**

| Octal | Meaning | TYPE |
|---|---|---|
| | HW dependant library | |
| 6201B | HW-Lib: Low-limit greater than high-limit in protection setting | ERROR |
| 6202B | HW-Lib: Attempt to prohibit R/W-access to master control register | ERROR |
| 6203B | HW-Lib: Attempt to read protection outside protected area | ERROR |
| 6204B | HW-Lib: Address does not match protection segment | ERROR |
| | OPCOM | |
| 6240B | Domino OPCOM: Invalid service request | ERROR |
| 6241B | Domino OPCOM: Exception occurred for which no handler exists<br>Octobus station.............. I20<br>Crate Id (MFB contr. station). I1D<br>MFB slot..................... I1D<br>Exception number............. I2H<br>Address where occurred........ I4H<br>I4Unused | FATAL |

*Table 10. DOMINO Services (HW-LIB/OPCOM) error codes*

**DOMINO Services (BOPCOM) error codes**

| Octal | Meaning | TYPE |
|---|---|---|
| | BOPCOM | |
| 6260B | BOPCOM : Server started<br>?? I2Unused<br>?? I2Unused<br>Version................... 34A | INFO |
| 6261B | BOPCOM: Path opened to controller<br>Octobus station........... I20<br>Message device............ I20 | INFO |
| 6262B | BOPCOM: Path released<br>Octobus station........... I20<br>Message device............ I20 | INFO |
| 6271B | BOPCOM: Too many Octobus errors | FATAL |
| 6272B | BOPCOM: Too many Superkernel errors | FATAL |
| 6273B | BOPCOM: Unable to open own superports<br>(XMSG not started?)<br>?? I2Unused<br>?? I2Unused<br>?? I20unused<br>?? I12unused<br>?? I2unused<br>External error............ SEC | FATAL |
| 6274B | BOPCOM: XMSG bufferspace exceeded | FATAL |

*Table 11. DOMINO Services (BOPCOM) error codes*

## NUCLEUS error codes

| Constant | Octal val | Meaning |
|---|---|---|
| nke_ERROR_BASE | 101000b | Base number for Nucleus errors |
| nke_ILLPAR | 101001b | Invalid parameter value |
| nke_ILLTYPE | 101002b | Wrong type used,- port, message or send reference |
| nke_NOMESS | 101003b | Both port and message in Send reference may not be zero |
| nke_ILLNO | 101004b | Port, message or send reference outside range |
| nke_NOTLOCAL | 101005b | Receive from remote port |
| nke_OUTSIDE | 101006b | Displacement outside buffer |
| nke_DESCARRFULL | 101007b | Descriptor table full |
| nke_BUFFULL | 101010b | Message buffer area full |
| nke_NAMEFULL | 101011b | Name table full |
| nke_NAMENOTFOUND | 101012b | Port name not defined |
| nke_NAMEUSED | 101013b | Port name already defined |
| nke_NOACCESS | 101014b | No access to given port, message or send reference |
| nke_ILLNETADDRESS | 101015b | Net address not found |
| nke_ILLKERNELNO | 101016b | Invalid kernel number |
| nke_NETTABFULL | 101017b | Net table full |
| nke_PROTOCERROR | 101020b | Inconsistent Nucleus module versions installed |
| nke_REJECTED | 101021b | Message rejected by receive process |
| nke_PORTNOTFOUND | 101022b | Port reference not defined in name server |
| nke_LOCK | 101023b | Unable to lock port |
| nke_NOTEVENBYTE | 101024b | Displacement not on even byte (only for ND-100) |

Continue on next page...

## NUCLEUS error codes

| Constant | Octal val | Explanation |
|---|---|---|
| nke_NOTINITIALISED | 101025b | Nucleus not started |
| nke_NAMEPORTUSED | 101026b | The Nameserver port is already initialised |
| nke_NAMEINDEXERROR | 101027b | Index error in Nameserver request |
| nke_INCONSISTENT | 101030b | Inconsistent structure in name server |
| nke_TOOMANYBYTES | 101031b | Buffer provided is too small |
| nke_PORTCLOSED | 101032b | Receive port is closed. |
| nke_ILLFUNC | 101033b | Invalid Function code |
| nke_PROTECTED | 101034b | Attempt to use protected Function |
| nke_ILLHARDWARE | 101035b | Not correct hardware configuration |
| nke_FATAL | 101036b | Fatal error in Nucleus |
| nke_QTABFULL | 101037b | Too many concurrent Nucleus users (quota table full) |
| nke_QUOTAUSED | 101040b | No more Nucleus resources available for this user |
| nke_ILLUSER | 101041b | Unknown user area identifier |
| nke_KICKLOCK | 101042b | Timeout when waiting for lock (kick-queue) |
| nke_DELAYTABFULL | 101043b | Unable to create more ports using delayed abort |
| nke_NOTAVAILABLE | 101044b | NUCLEUS not available in CPU. (not started or stopped) |
| nke_ILLVERSION | 101045b | Invalid version of NUCLEUS library |

**NUCLEUS calls and error codes**

| | nkCrePort | nkCreName | nkOpenPort | nkOpenReturnPort | nkDelName | nkCreMessage | nkMove | nkSend | nkReceive | nkClose | nkGetInfo | nkVersion |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| nke_BUFFULL | | | | | | X | | | | | | |
| nke_DELAYTABFULL | X | | | | | | | | | | | |
| nke_DESCARRFULL | X | | | | | X | | | | | | |
| nke_FATAL | X | X | X | X | X | X | | | | X | | |
| nke_ILLFUNC | X | X | X | X | X | X | X | X | X | X | X | X |
| nke_ILLHARDWARE | | | | | | | | | | | | |
| nke_ILLKERNELNO | | | | | | | | | | | | |
| nke_ILLNETADDRESS | | | | | | | | | | | | |
| nke_ILLNO | | X | X | X | X | X | X | X | X | X | X | |
| nke_ILLPAR | | X | | | | | | | | | | X |
| nke_ILLTYPE | | X | | X | X | X | X | X | X | | X | |
| nke_ILLUSER | X | | | | | X | | | | | | |
| nke_ILLVERSION | X | X | X | X | X | X | X | X | X | X | X | X |
| nke_INCONSISTENT | | | | | | | | | | | | |
| nke_KICKLOCK | | | | | | | | | | | | |
| nke_LOCK | | | | | | | | | | | | |
| nke_NAMEFULL | | X | | | | | | | | | | |
| nke_NAMEINDEXERROR | | | | | | | | | | | | |
| nke_NAMENOTFOUND | | | X | | X | | | | | | | |
| nke_NAMEPORTUSED | | | | | | | | | | | | |
| nke_NAMEUSED | | X | | | | | | | | | | |
| nke_NETTABFULL | | | | | | | | | | | | |
| nke_NOACCESS | | | X | X | | X | X | X | X | X | X | |
| nke_NOMESS | | | | | | | | X | | | | |
| nke_NOTAVAILABLE | | | | | | | | | | | | X |
| nke_NOTEVENBYTE | | | | | | | X | | | | | |
| nke_NOTINITIALISED | X | X | X | X | X | X | X | X | X | X | X | X |
| nke_NOTLOCAL | | | | | | | | | | | | |
| nke_OUTSIDE | | | | | | X | X | | | | | |
| nke_PORTCLOSED | | | | | | | | X | | | | |
| nke_PORTNOTFOUND | | | | | | | | | | | | |
| nke_PROTECTED | | X | | | | | | | | | | |
| nke_PROTOCERROR | | | | | | | | | | | | |
| nke_QTABFULL | | | | | | | | | | | | |
| nke_QUOTAUSED | X | | | | | X | | | | | | |
| nke_REJECTED | | | | | | | | | X | | | |
| nke_TOOMANYBYTES | | | | | | | | | | | X | X |

## NUCLEUS operation error/status codes

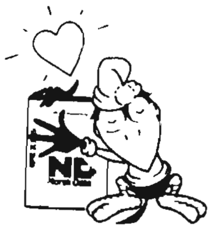
| Octal | Meaning | TYPE |
|---|---|---|
| 1011b | Nucleus Operation | |
| 101100b | Nucleus Name server started<br>Version....:   34A<br>Size of name table..:  I40 | INFO |
| 101101b | Nucleus server started<br>Version....:   34A<br>Cluster Id (ND-100 Octobus station no. I40<br>Zeropage for multiport memory......... I40<br>Page start address of kernel within<br>multiport memory...................... I40<br>Nucleus kernel size in pages.......... I40<br>Message buffer space for all<br>system processes in pages............ I40<br>Number of descriptors for all<br>system processes in pages............ I40<br>Message buffer space for all<br>public processes in pages............ I40<br>Number of descriptors for all<br>public processes...................... I40<br>Message buffer per public process<br>in pages.............................. I40<br>Number of descriptors per public<br>process............................... I40<br>Trace buffer space in pages........... I40 | INFO |

Continue on next page....

| Octal | Meaning | TYPE |
|---|---|---|
| 1011b | Nucleus Operation | |
| 101102b | Unable to start server<br>Nucleus error........................ SEC<br>Sintran error........................ SEC | FATAL |
| 101103b | Name server stopped<br>Nucleus error........................ SEC | FATAL |
| 101104b | Unable to reserve mailbox | FATAL |
| 101105b | Unable to get Cpu type | FATAL |
| 101106b | Unable to get Nucleus configuration from Sintran | FATAL |
| 101107b | Unable to get start address of multiport | FATAL |
| 101110b | Unable to find own Octobus station | FATAL |
| 101111b | Unable to fix memory for Nucleus | FATAL |
| 101112b | Unable to initialise Nucleus kernel | FATAL |
| 101113b | Unable to initialise Sintran part of Nucleus | FATAL |
| 101114b | Unable to connect to Octobus | FATAL |
| 101115b | Inconsistent data structure<br>Detected at address.................. I40<br>Called from.......................... I40 | FATAL |
| 101116b | Timeout when waiting for lock<br>Called from.......................... I40<br>Address of lock...................... I40 | FATAL |
| 101117b | Nucleus may not be restarted, please restart system | FATAL |
| 101120b | Unable to find Nucleus kernel | FATAL |

*Table 12. NUCLEUS operation error/status codes*

# Index




## SEND US YOUR COMMENTS!

Are you frustrated because of unclear information in our manuals? Do you have trouble finding things?

Please let us know if you:
- find errors
- cannot understand information
- cannot find information
- find needless information.

Do you think we could improve our manuals by rearranging the contents? You could also tell us if you like the manual.

Send to:
Norsk Data A.S
Documentation Department
P.O. Box 25 BOGERUD
N - 0621 OSLO 6 - Norway

## NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Manual Name: ______________________ Manual number: ____________

Which version of the product are you using? ______________________

What problems do you have? (use extra pages if needed) ______________________

Do you have suggestions for improving this manual? ______________________

Your name: ______________________ Date: ____________

Company: ______________________ Position: ____________

Address: ______________________

What are you using this manual for? ______________________