EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

PS/CO/NOTE 87-011
28 October 1987

Project: COMMUNICATIO
Domain: DATAGRAM
Category: REFMANUAL
Status: DRAFT

DATAGRAM SERVICE FOR CONSOLE, FEC AND SMACC

REFERENCE MANUAL

Alain Gagnaire

Abstract

This note presents the implementation and the user interface of the CERN internetwork datagram protocol implemented on ND100 (FEC, CONSOLE) and SMACC's

It provides users with:
- the functionality of the datagram protocol.
- a description of the user interface
- a programmer's guide for: NODAL, compiled languages, the M68K assembler
- an implementation note to the current implementation

Geneva, Switzerland

TABLE OF CONTENTS

Section		Page
1	INTRODUCTION	1
2	Main features of the datagram protocol	2
2.1	Service provided	2
2.2	What is not provided	2
2.3	Network layer Addressing	3
3	Functionality of the datagram service	4
3.1	Formal Definition of the data structure type	4
3.1.1	Internet address	4
3.1.2	Socket information	4
3.2	The datagram service user interface: procedure call	4
3.2.1	Opening a socket activity	4
3.2.2	Setting socket characteristics	5
3.2.3	Closing socket activity	5
3.2.4	Sending a datagram	6
3.2.5	Receiving a datagram	6
3.2.6	Waiting for internet events	6
3.2.7	Getting socket information	7
3.2.8	Getting the list of currently opened socket	7
3.2.9	Closing a target socket	7
4	Allocation of socket number:	8
5	Type of access to user interface	8
6	Event for synchronisation	8
6.1	Synchronisation on ND100:	8
6.2	Synchronisation on SMACC:	9
7	NODAL interface	10
7.1	NODAL representation of structures	10
7.2	Nodal syntax of common functions:	10
7.2.1	DGOPN:	10
7.2.2	DGCLS:	10
7.2.3	DGSND:	11
7.2.4	DGRCV	11
7.2.5	DGINF:	12

Section Page

7.3 Nodal implementation specific to ND100 12
7.3.1 DGWT: 13
7.4 Nodal implementation specific to the SMACC 13
7.4.1 DGENB: 13
7.4.2 DGDIS: 14
7.4.3 Processing the DG Event from the ASQ 14
7.5 NODAL example of synchronisation with DG service 14

8 User interface for high-level languages 16

8.1 PPL interface module for DG service: 16
8.2 C interface for DG service on SMACC 19

9 SMACC assembly language interface to DG service 21

9.3 Description of Trap server interface for DG service 21
9.4 Directives number of DG trap server 21
9.5 Parameter block of directives 21
9.5.1 DGOPN parameter block: 22
9.5.2 DGCLS parameter block 22
9.5.3 DGSND parameter block 22
9.5.4 DGRCV parameter block 22
9.5.5 DGSET parameter block 22
9.5.6 DGINF parameter block 23
9.5.7 DGENB parameter block 23
9.5.8 DGDIS parameter block 23
9.5.9 Structure of event 23
9.5.10 Mask for event 23
9.5.11 Error code 24

APPENDIX

A Implementation Note 25

10 Implementation remarks: 26
10.1 What is not implemented 26
10.2 Data structure in DG service 26
10.2.1 Configuration parameter of datagram service: 27
10.2.2 Parameters defined at compiled time: 27
10.2.2.1 Parameter defined at load time: 28

B Reminder of basic definitions 29

Section Page

11 Wait_mask structure: 30
11.1 Network address structure: 30
11.2 Allocation of socket number: 30
11.3 Socket information 31
11.4 Error code given back by datagram service 31

1 INTRODUCTION

This note is the Programmer's guide of the datagram service, implemented on:

-FECs and CONSOLEs, i.e. ND100 computers, running the Sintran III operating system, and using the PS/CO implementation of the TITN network to provide a link level transmission service.

- SMACCs controlled via a serial Camac loop by a FEC. The link level protocol is provided by the datagram service running on the FEC, based on a new version of the serial LAM handling.

Consequently, this implementation allows point to point communication between user on any computer of the PS control system, i.e.:

- *SMACC to SMACC on the same FEC*
- *SMACC to SMACC on a different FEC*
- *SMACC to FEC*
- *SMACC to CONSOLE*
- *FEC to SMACC*

and between CONSOLEs and FECs

The implementation of the datagram service is based on the definition given by the interim report <1>

However This implementation does not conform to all of the Interim report (see appendix A on implementation remarks).

<1> Interim report of the Datagram Definition Group,
(DD/KIK- X/R1/W1/W9/G2 C. Piney, B. Carpenter, Keyser)

2 Main features of the datagram protocol

The datagram service defined by the interim report is a connectionless network service.

2.1 Service provided

- The basic service is the transmission of a block of data (datagram) from a sender to a receiver.
- The datagram service (DG) will accept and transmit datagram whose maximum byte size depends on the ressources available along the way between source and destination.

- If, for any reason, the datagram is too large for complete delivery, it will be truncated and the receiver informed.

- The DG will transmit , together with the data,a header made of parameters for its own use. Corruptions of the DG's header during transmission will cause the destruction of the datagram,and this guaranteedit not to be delivered to the wrong user.
- Included in the DG's parameters will be the source of the datagram which will be available to the receiving client.

- The DG will provide the necessary routing of the datagram between DG nodes

- There will be a two level prority scheme for the transmission of datagrams. Those with the higher priority will be queued for transmission before those of lower priority. However, a lower priority datagram in the process of transmission will not be interrupted in order to transmit a higher priority one.

2.2 What is not provided

- The service will make no provision for flow control.

- The service will not guarantee delivery of the datagram, nor will it provide
to the sender an acknowledgement of delivery.

- The service cannot guarantee the non-duplication of a datagram.

- The service does not guarantee that the order of receiving datagrams will be that of their sending.

2.3 Network layer Addressing

- The Medium Access Controladdress is a 48 bits field split in 3 equal part of 16 bits,corresponding to the ISO proposal.

- The final entity addressed through the network layer is a 16 bit Network Service Access Point selector, called in our environment a Socket.

- The complete network address is the concatenation of the medium access control address (MAC) with the socket address.

The structure of the whole network address is:

```
% 16 bits %   ISO_code
% 16 bits %   network_number
% 16 bits %   station_number
% 16 bits %   socket
```

Where:

ISO_code =
- bit 1 = I/G to distinguish Individual address from Group address
  0 for Individual, 1 for Group.
  use always 0.
- bit 0 = L/U to distinguish locally from universally administered address.
  use always 1.
- bit 2..15 = unused bits, always= 0

network_number= to identify a physical network or subnetwork.
use the value 0.

station_number= to identify the number of the station in the network.

If the addressed station is a FEC, it is the TITN line number.
If the addressed station is a SMACC, the structure of station_number is:
- 8 MS bits= FEC line value
- 8 LS bits= SMACC number

Socket_number= has the role of NetWork Service Access Point (NSAP) selector, identifying the user in the net work

N.B.:
Since 0 is not allowed as value of a local station the SMACC numerotation should start with 1. <1>

<1> System software layout for MC68000 super ACC, (PS/CO Note 83-015 B. Carpenter,G. Cuisinier,A. Gagnaire,K. Osen,W. Remmer)

3 Functionality of the datagram service

3.1 Formal Definition of the data structure type

The PPL notation is used for the formal description.

3.1.1 Internet address

This is defined by the record:

```
TYPE internet_address:
     RECORD
        ISO_code:        INTEGER      % 16 bits: holds the I/G, L/U bits
        network_number:  INTEGER      % 16 bits: identifies the LAN
        station_number:  INTEGER      % 16 bits: identifies the host
        socket:          INTEGER      % 16 bits: identifies the NSAP
     END RECORD
```

3.1.2 Socket information

```
TYPE socket_info:
     RECORD
        local:              internet_address
        filter:             internet_address
        priority:           BOOLEAN     % 16 bits
        discard_timeout:    INTEGER     % 16 bits
        inqueue:            INTEGER     % 16 bits
        outqueue:           INTEGER     % 16 bits
        credit:             INTEGER     % 16 bits
     END RECORD
```

3.2 The datagram service user interface: procedure call

The NODAL syntax is used to present the procedure call.

3.2.1 Opening a socket activity

DG_OPEN(socket,access_number,status)

This informs the datagram service that the calling task will exchange datagrams with the internet, and that it will be addressed by the given socket number. Th specified socket number is a 16 bit positive value, except the value 0 which is used to request a dynamic allocation of the socket number, in this case a value is allocated in the negative ranges: [8001..80FF] hexa. It allows to use the datagram service in calling mode without specifying a predetermined socket number.

The datagram service returns a local access number, the default setting of the parameters controlled by the DGSET primitives apply.

There is no communication with the internet.
An open socket enables the network service access point: it can receive datagrams from outside, stand user calls to send/receive datagram.
The status returns the completion code:

If status =0 the socket is open and AN gives the access number to use in subsequent calls
ifstatus = DG_already_open, this informs the caller that the socket is already opened by him and AN gives the access number, otherwise the error code gives the reason of the unsuccessfull Open.

3.2.2 Setting socket characteristics

DG_SET(access_number,filter,priority,discard_timeout,credit_request,status)

Sets up a socket if non-default values are required:

filter: is an internet_address
from now onwards, datagram are to be accepted only from the given source.
Default value= ALL

priority: if TRUE, specifies priority transmission.
Default value= FALSE

discard_timeout: specifies how many seconds to keep an incoming datagram before discarding it, if no receive call is made to the socket concerned.
Default= 10

credit_request: requests a new credit limit in Kbytes for the datagrams instantaneously queued on behalf of the socket concerned.
Default= depends on configuration of service (room dedicated to buffer management, nb of socket and datagram declared).

3.2.3 Closing socket activity

DG_CLOSE(access_number,status)

Informs the datagram service that the socket is no longer needed.
If access_number =0 all the sockets opened by the caller are closed

3.2.4 Sending a datagram

DG_SEND(access_number,byte_count,data,destination,immediate,status)

The data are queued for transmission through the internet and the user buffer is free for re-use (non blocking call). The destination is addressed by an internetwork_address.

immediate: Boolean if TRUE, the datagram is considered to be complete and immediately queued for transmission.
if FALSE, the datagram is considered to be incomplete, and subsequent calls to DG_SEND will cause data to be appended to the end of the current datagram. The final data are provided by a call with immediate set TRUE, the datagram is then considered to be completed and is immediately queued for transmission.
The call is non-blocking (no implicit wait).

3.2.5 Receiving a datagram

DG_RECEIVE(access_number,buffer_sizer,byte_count,data,source,status)

- *If an incoming datagram is waiting at the specified socket, and if it is not bigger than the indicated buffer_size in bytes, it is returned to the buffer, with byte_count indicating its size in bytes.*
  Otherwise an appropriate status code is returned.

- *If a datagram is received which is too big for the buffer, a status code indicating truncation is returned.In this case the buffer is filled with the first buffer_size bytes of the datagram (starting at the beginning). The byte_count parameter indicates the number of bytes in the datagram remaining unread.*
  Further calls (with similar return parameter) may then be made to DG_RECEIVE , in order to receive subsequent parts of the datagram in a sequential manner.
  The end of the datagram is signalled by an OK status code with byte_count indicating the actual number of bytes in the final part of the datagram
  The call is non-blocking (no implicit wait).

3.2.6 Waiting for internet events

Provides the user with means to be synchronised with network and datagram service events.The general form of the primitive is represented by the call:

DG_WAIT(access_number,wait_mask,system_mask,
timeout,return_mask,which_socket, status)

But due to implementation constraints this primitive is not implemented in the same manner in the ND100 and in the SMACC.

Anyway the principle is unique: the user has to wait until one of the events, he has selected, arises. The sources of events are:

- *the datagram service (specified by wait_mask)*
- *the outside world (specified by system_mask)*
- *the clock (specified by timeout)*

The coding of system_mask is undefined and system dependent and not supported at present.

3.2.7 Getting socket information

DG_INFO(access_number,current_state,status)

Obtains the information about the socket in the current_state parameter of type socket_info.The information given is:

local	=>	*internet address of local socket*
filter		
priority	=>	*socket_info area to get current state*
discard_timeout		*set up with DG_SET*
credit_request		
inqueue	=>	*number of incoming datagram waiting*
outqueue	=>	*number of outgoing datagram waiting*
credit	=>	*current free space for datagram in Kbytes*

3.2.8 Getting the list of currently opened socket

DG_LIST

List the open socket on user device, giving name of the owner, called User_name

3.2.9 Closing a target socket

DG_PCLOSE(user_name,socket,returned_value,status)

Forces the socket owned by user_name to be closed.

If the name is an empty string the calling user is assumed.
If the socket number is 0, all sockets of User_name are to be closed.

4 Allocation of socket number:

Some socket numbers are going to be the identification of a service and that independantly of the host. To normalise the use of the socket number and to guarantee their uniqueness, it is proposed to define 3 ranges of use:

- *Socket identifying services: number 1..255 (FF hexa)*
- *Socket identifying application: number >= 256 (100 hexa)*

To access services the calling mode does not need a fixed allocation of the socket, in that case a dynamic allocation will be performed:

- *Socket dynamically allocated: negative number.*

The dynamic allocation of socket is obtained by specifying 0 as socket number in DGOPN.

5 Type of access to user interface

The set of primitives are provided for ND100 and M68K SMACC:

On the ND100 CONSOLE or FEC, they can be called:
- *from RT NODAL,*
- *as ICCI function for Real Time user program in PPL*

On the SMACC, they can be called:
- *directly via their low level visibility: the Trap server and its convention*
- *from NODAL*
- *from compiled programs using CERN standards with the library providing the call to the trap server interface*

6 Event for synchronisation

The user of the DG service wants to be synchronised with events occurring in the network in order to control what happens on the local station and to manage the application according to these events. The synchronisation is system-dependent, this is why implementation is different on the SMACC and the ND100.

6.1 Synchronisation on ND100:

In one call (DGWT) the user expresses the event he wants to receive and enters the wait. While waiting for a DG event, the RT of the user can be exited from that status by other reasons. Due to Sintran III, the DG user is informed by the returned value :

if value = 0 , dg_ok: event occurred look at R_MSK to know which kind,
-1, dg_timeout: time given for waiting elapsed,

-2, dg_prompted: removed from time queue while waiting,
-3, dg_didnotwait: no wait done becauseof the bit repeat which made the wait dummy.

The returned wait mask gives the origin of the event according to previous event mask specifications.

6.2 Synchronisation on SMACC:

With one function DGENB, the user expresses the events he wants to be synchronised with.
With his ASQ, he can then get synchronised with events coming from the ASQ. The events generated by the DG service on the ASQ of the user have a dedicated structure:

```
% one byte %              Event size =    $8
% one byte %              Type =          $22
% one 16 bits word   %    Identificator=  $4447= 'DG'
% one 16 bits word   %    Return_mask =   mask specifying
                                          which event occured
% one 16 bits word   %    Which_socket =  socket number
                                           where event occured
```

7 NODAL interface

The NODAL functions perform the service described above. Due to implementation constraints we have one subset common to any implementation and one other another to the local implementation, solving the synchronisation of service.

7.1 NODAL representation of structures

The data structures are made up with declarations of integer arrays:

```
>DIM-INT NW.AD(4)    % - for network_address 4 integers at least:
>DIM-INT SK.INF(13)  % - for socket_info 13 integers at least:
>DIM-INT BUF(?)      % - for data buffer integer array whose size
                         depend on datagram size and local available
                         room To minimize the overhead, the buffer size
```

should be >= datagram size

The other paramaters are standard NODAL variables.

7.2 Nodal syntax of common functions:

These functions are the same on ND100 and SMACC

7.2.1 DGOPN:

To Open a socket:
>DGOPN(SK,AN,CC)

SK: variable, defining the socket number to be opened
in the range 1..32768 (7FFF hexa).

0 is not a socket number, it tells the DG service to dynamically allocate a socket number(in the range 8001..80FF hexa).

AN: variable, receiving access_number required for further use of the socket.
CC: variable, giving back the status
0 if OK and AN is meaningful.

7.2.2 DGCLS:

To close a socket:
>DGCLS(AN,CC)

AN: variable defining the access-number of the socket to be closed

CC: variable, giving back the status

To close all sockets of the caller, one has to give 0 as access number, e.g.:>DGCLS(0,CC)

7.2.3 DGSND:

To send a datagram to the destination given in DEST:

> DGSND(AN,BC,BUF,DEST,IM.,CC)

AN : variable specifying the access_number of target socket,
BC : variable giving the actual size to be sent, in bytes,
BUF: integer array giving the source of data to be sent,
DEST: Array of 4 integer giving the network destination address,
IM.: variable for immediate flag
CC: variable, giving back the status

Example:

```
>DIM-INT BUF(100)        % array for data to be send
>DIM-INT DEST(4)               % array to define dest network add
>SET SK=7                      % socket number of dest user
>SET DEST(1)=2;                % dest ISO_code always 2
>SET DEST(2)=0;                % dest network number 0 in our context
>SET DEST(3)=SYMBOL("BENCH")   % dest station
>SET DEST(4)=7                 % dest socket of receiver


>SET BC=50                     % actual number of bytes to be sent
>SET IM.=1                     % Immediate send flag set
>DGSND(AN,SZ,BUF,DEST,IM.,CC)
```

7.2.4 DGRCV

To receive a datagram from a socket, the network source address is given by SRCE

> DGRCV(AN,SZ,BC,BUF,SRCE,CC)

AN : variable specifying the access_number of target socket
SZ : variable specifying the actual size of buffer
BC: variable giving back info on datagram currently read:
if datagram is read out (CC =0) it gives the number of bytes received.
if datagram is truncated (CC= DG_truncated) it gives the number ofbytesremainigto be read and SZ is the number of bytes picked up from the datagram in BUF.
BUF: integer array to read data from datagram.
SRCE: array of 4 integers receiving network source address of the datagram sender
CC : status given back

Example:

```
>DIM-INT BUF(100)        % array for data to be sent
>DIM-INT  SRCE(4)              % array to get network source add
                               % source add


>SET SZ=ARSIZE(BUF)*2          % actual size of buffer in byte
>SET BC=0                      % to get byte_count
>DGRCV(AN,SZ,BC,BUF,SRCE,CC)
                               % SRCE(1) = source ISO_code of sender
                               % SRCE(2) = source network of sender
                               % SRCE(3) = source station of sender
                               % SRCE(4) = source socket of sender
```

7.2.5 DGINF:

To get socket information:

```
>DGINF(AN,SK.INF,CC)
```

AN : variable specifying the access_number of the target socket.
SK.INF: Array of 13 integers returning socket information.
CC : status given back

Example of sequence:

```
>DIM-INT SK.INF(13)
>  .
>DGINF(AN,SK.INF,CC)
>  .       % SK.INF(1) = local network add: ISO_code
           % SK.INF(2) =           "       : network_number
           % SK.INF(3) =           "       : station_number
           % SK.INF(4) =           "       : socket_number
           % SK.INF(5) = network address filter for: ISO_code
           %SK.INF(6) = "                 : network_number
           % SK.INF(7) = "                 : station_number
           % SK.INF(8) = "                 : socket_number
           % SK.INF(9) = priority: boolean 0 => FALSE, 1 => TRUE
           % SK.INF(10)= discard timeout value
           % SK.INF(11)= nb of datagram of inqueue
           % SK.INF(12)= nb of datagram of outqueue
           % SK.INF(13)= credit value in Kbyte
```

7.3 Nodal implementation specific to ND100

These functions are implemented in a different manner on ND100 and SMACC.

7.3.1 DGWT:

To synchronise the user with datagram events:

```
>DGWT[AN,W.MSK,S.MSK,TO,R.MSK,SKT.,CC]
```

AN : variable specifying the access_number of target socket
W.MSK : variable specifying the mask whose 16 bits, according to the specification, select the kind of wait required.
S.MSK : variable ignored, option not implemented today.
TO : variable giving the value of time-out to control waiting.
R.MSK : variable getting the value of mask which tells the reason of wakeup.
SKT. : variable returning the access number for which the significant event occurred (only useful when bit 0 of wait_mask is set).
CC : variable: returning the reason of leaving the wait. The interpretation of returned code is found in Appendix B.

Example:

```
                        % set up mask to select mode of wait
>SET W.MSK=0
>SET BIT[1,W.MSK]=1 % sets the bit time_bit=1
>SET BIT[3,W.MSK]=1 % sets the bit inq_bit=3
>SET TO= 10         % ask 10 seconds to wait
>DGWT[AN,W.MSK,S.MSK,TO,R.MSK,SKT.,CC]
                        % BIT[1,R.MSK]=1 if time-out occurred
                        % BIT[3,R.MSK]=1 if datagram in inqueue
```

7.4 Nodal implementation specific to the SMACC

Due to implementation constraint, the synchronisation with network events on the SMACC, running under the control of the RMS monitor, is done in two steps:

- *firstly, selection of requested event (setting mask in socket) in order to tell the DG service to generate an event on the ASQ when one of the selected events occurred.*
- *secondly wait for ASQ, with the RMS command.*

7.4.1 DGENB:

To ask the DG service to issue an event on the task's ASQ when a requested event arrived.

```
>DGENB[AN,W.MSK,CC]
```

AN : variable specifying the access_number of target socket
W.MSK : variable specifying the mask whose 16 bits, according to the specification, will enable event generation when the conditions specified by the mask are fullfilled,

CC : status given back

7.4.2 DGDIS:

To stop the activity requested by DGENB only for the specified mask.

```
>DGDIS(AN,W.MSK,CC)
```

AN : variable specifying the access_number of target socket
W.MSK : variable whose 16 bits according to specification, will disable generation of events for the conditions specified by the mask.
CC : status given back

7.4.3 Processing the DG Event from the ASQ

refer to RMS user's guide to know all about RMS synchronisation processing

To Wait and get an event from the ASQ, use the Nodal function :

```
>∂GET-AN-EVENT,MSG.;
```

MSG.: array of integer to receive the event.

```
MSG.(1)= Size and type = $0822
MSG.(2)= Identificator = $4447='DG'
MSG.(3)= Ret_mask
MSG.(4)= Which_socket
```

7.5 NODAL example of synchronisation with DG service

1) Allocate ASQ for the task:

```
>DIM-INT MSG.(16)  % to stand any size of standard event
>∂ALLOCATE-A-S-Q,,,[41,16,8*16,0;
```

2) to set up time-out, request periodic activation

```
>∂REQ-PER-ACT,,,[44000,0,60000,,,;   % 1 minute time out
```

3) select event to receive:

```
>SET W.MSK=8;                % select inqueue non empty.
>DGENB(AN,W.MSK,CC)          % Enable acceptance of the event.
```

4) wait and recognise the event

```
>∂GET-AN-EV,MSG.;
```

```
>IF MSG.(2)><[[4447; DO xx; % not a DG event
>DGRCV(AN,SZ,BC,BUF,SRCE,CC) % Read the incoming datagram
```

8 User interface for high-level languages

8.1 PPL interface module for DG service:

Text of the :DEFN file providing the module giving syntax rules to call datagram service:

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %                                                        %
% %      SOURCE FILE : (CONSOLE-RN500)DG-USER-INT:DEFN     %
% %                                                        %
% % VERSION 1 :                                            %
% %     1ST EDIT  = 08-OCT-1985   Alain Gagnaire           %
% %     LAST EDIT = 07-MAY-1986   Fabien Perriollat        %
% %                                                        %
% %                                                        %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %                                                        %
% %                                                        %
% %     User definition module of the PS datagram service %
% %                                                        %
% %     For general information see the document of the    %
% %    DG  datagram study group of CERN.                   %
% %                                                        %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       MODULE PROVIDING DG_USER_INT ON DGSEG:

%  PROVIDE ALL WITH
%
%  (* Datagram service specific types and constants *)
%  (* -------------------------------------------- *)

%  (* Format of datagram MASK words
%   ------------------------------
%   bit set to 1 ==>
%      bit 0 = applies to all sockets opened by this user
%               (access number ingnored)
%      bit 1 = timeout required
%      bit 2 = outside world wait (system mask used)
%      bit 3 = wait till inqueue non-empty
%      bit 4 = wait till outqueue empty
%      bit 5 = wait till credit >= credit request
%  *)
%
%  CONSTANT
%
%  (* user interface error and warning codes for status returns *)
%
```

```
%          dg_OK:                      0;
%          dg_queue_empty:             1;
%          dg_no_credit:               2;
%          dg_invalid_access_number:   3;
%          dg_invalid_credit_request:  4;
%          dg_invalid_timeout:         5;
%          dg_internal_error:          6;
%          dg_truncated:               7;
%          dg_already_open:            8;
%          dg_buffer_size_error:       9;
%          dg_not_started:            10;
%          dg_data_empty:             11;
%          dg_invalid_parameter:      12;
%
%       (* warning codes specific to dg_wait *)
%          dg_timeout:    -1; (* time given for waiting elapsed,
%                              M267 status=0 *)
%          dg_prompted:   -2; (* removed from t.q. while waiting,
%                              M267 status=1 *)
%          dg_didnotwait: -3; (* repeat bit made the wait dummy,
%                              M267 status=-1 *)
% (**)
%
% (* specific types *)
%
% TYPE internet_address:     (* CERN/ISO internet address format *)
%       RECORD
%          ISO_code:       INTEGER; (* I/G bit, L bit, 14 zero bits *)
%          network_number: INTEGER; (* 16 bits identifying LAN *)
%          station_number: INTEGER; (* 16 bits identifying computer *)
%          socket:         INTEGER  (* 16 useful bits *)
%       END RECORD;
%
%
% TYPE socket_info:          (* used to return information to user *)
%             RECORD
%               local:            internet_address;
%               filter:           internet_address;
%               priority:         BOOLEAN;
%               discard_timeout:  INTEGER;
%               inqueue:          INTEGER;
%               outqueue:         INTEGER;
%               credit:           INTEGER
%             END RECORD;
%
% #PAGE#
% (* Datagram service primitive procedures *)
% (* ------------------------------------ *)
%
% PROCEDURE dg_open    ( RO socket:        INTEGER;
%                        WO access_number: INTEGER;
%                        WO status:        INTEGER)
%           ENTRY 'DGOPN' AS ICCI;
%
% PROCEDURE dg_set     ( RO access_number: INTEGER;
%                        RO filter:        internet_address;
```

```
%                          RO priority:       BOOLEAN;
%                          RO discard_timeout: INTEGER;
%                          RO credit_request:INTEGER;
%                          WO status:         INTEGER)
%           ENTRY 'DGSET' AS ICCI;
%
% PROCEDURE dg_close    ( RO access_number: INTEGER;
%                          WO status:         INTEGER)
%           ENTRY 'DGCLS' AS ICCI;
%
% PROCEDURE dg_send     ( RO access_number: INTEGER;
%                          RO byte_count:     INTEGER;
%                          RO data:           ROW [LO..HI:INTEGER] OF INTEGER;
%                          RO destination:    internet_address;
%                          RO immediate:      BOOLEAN;
%                          WO status:         INTEGER)
%           ENTRY 'DGSND' AS ICCI;
%
% PROCEDURE dg_receive  ( RO access_number: INTEGER;
%                          RO buffer_size:    INTEGER;
%                          WO byte_count:     INTEGER;
%                          RW data:           ROW [LO..HI:INTEGER] OF INTEGER;
%                          WO source:         internet_address;
%                           WO status:         INTEGER)
%           ENTRY 'DGRCV' AS ICCI;
%
% PROCEDURE dg_wait     ( RO access_number: INTEGER;
%                          RO wait_mask:      INTEGER;
%                          RO system_mask:    INTEGER;
%                          RO timeout:        INTEGER;
%                          WO return_mask:    INTEGER;
%                          WO which_socket:   INTEGER;
%                          WO status:         INTEGER)
%           ENTRY 'DGWT' AS ICCI;
%
% PROCEDURE dg_info     ( RO access_number: INTEGER;
%                          WO current_state: socket_info;
%                          WO status:         INTEGER)
%           ENTRY 'DGINF' AS ICCI;
% #PAGE#
% END DG_USER_INT.
```

8.2 C interface for DG service on SMACC

```
% /* dg.h datagram include file */

% #define DGEVNT 0x22
% #define DGID 0x4447

% typedef struct {
%         short ISO_code;
%         short network;
%         short station;
%         short socket;
%         } dg_add;
%
%typedef struct {
%         dg_add local;
%         dg_add filter;
%         short priority;
%         short discard_timeout;
%         short inqueue;
%         short outqueue;
%         short credit;
%         } skt_info;
%
%/* parameter blocks */
%
%typedef struct {
%         short socket;
%         short access;
%         } pDGOPN;
%
%typedef struct {
%         short access;
%         } pDGCLS;
%
%typedef struct {
%         short access;
%         short count;
%         char *data;
%         dg_add *dest;
%         short flag;
%         } pDGSND;
%
%typedef struct {
%         short access;
%         short buffSize;
%         short count;
%         char *data;
%         dg_add *source;
%         } pDGRCV;
%
%typedef struct {
```

```
%        short access;
%        dg_add *filter;
%        short priority;
%        short timeOut;
%        short credit;
%        } pDGSET;
%
%typedef struct {
%        short access;
%        skt_info *state;
%        } pDGINF;
%
%typedef struct {
%        short access;
%        short mask;
%        } pDGENB;
%
%typedef struct {
%        short access;
%        short mask;
%        } pDGDIS;
%
%extern long DGOPN();
%extern long DGCLS();
%extern long DGSND();
%extern long DGRCV();
%extern long DGSET();
%extern long DGINF();
%extern long DGENB();
%extern long DGDIS();
%
%/*  Actual syntax:
%
%long DGOPN(parameter_block)
%        pDGOPN *parameter_block;
%long DGCLS(parameter_block)
%        pDGCLS *parameter_block;
%long DGSND(parameter_block)
%        pDGSND *parameter_block;
%long DGRCV(parameter_block)
%        pDGRCV *parameter_block;
%long DGSET(parameter_block)
%        pDGSET *parameter_block;
%long DGINF(parameter_block)
%         pDGINF *parameter_block;
%long DGENB(parameter_block)
%          pDGENB *parameter_block;
%long DGDIS(parameter_block)
%          pDGDIS *parameter_block;
%*/
```

9 SMACC assembly language interface to DG service

The DG service is implemented as a trap server, so the direct interface to it is the description of the call of a trap serverrespecting the convention of parameters passing.

9.3 Description of Trap server interface for DG service

On the SMACC the DG service is implemented as a trap #6 server whose calling sequence is:

```
MOVEQ     #directive_number,D0
LEA       Parameter_block,A0
TRAP      #6
BNE       TrapError
```

The standard for trap error return code is:

The register D0 gives back the error return code whose structure is:

```
          b1b2b3b4
Where
b1 =    Trap number in the 4 ms bits ( trap_number *8).
b2 =    directive number.
b3b4 =  error code returned. The register A0 is not affected.
```

9.4 Directives number of DG trap server

The invocation of the trap server selects the primitive with the directive number:

1 for Opening a socket
2 " Closing a socket
3 " Sending a datagram
4 " receiving a datagram
5 " not implemented
6 " setting a socket
7 " getting socket information
8 " Selecting and Enabling event
9 " Disabling of event selection

9.5 Parameter block of directives

By calling trap server a parameter block is passed whose structure is specific of the directive invoked:

9.5.1 *DGOPN parameter block:*

Directive number= 1

byte disp:	volume	Content
0	16 bit integer	Socket number
2	"	Access_number

9.5.2 *DGCLS parameter block*

Directive number= 2

byte disp:	volume:	Content:
0	16 bit integer	Access_number

9.5.3 *DGSND parameter block*

Directive number= 3

byte disp:	volume:	Content:
0	16 bit integer	Access_number
2	16 bit integer	byte_count
4	32 bit integer	data reference
8	32 bit integer	destination : Internet_address reference
12	16 bit integer	flag immediate

9.5.4 *DGRCV parameter block*

Directive number= 4

byte disp:	volume:	Content:
0	16 bit integer	Access_number
2	16 bit integer	buffer_size
4	16 bit integer	byte_count
6	32 bit integer	data reference
10	32 bit integer	source: Internet_address reference

9.5.5 *DGSET parameter block*

Directive number= 6

byte disp:	volume:	Content:
0	16 bit integer	Access_number
2	32 bit integer	filter: Internet_address reference
6	16 bit integer	priority
8	16 bit integer	discard timeout
10	16 bit integer	credit request

9.5.6 DGINF parameter block

Directive number= 7

byte disp:	volume:	Content:
0	16 bit integer	Access_number
2	32 bit integer	current_state: socket_info reference

9.5.7 DGENB parameter block

Directive number= 8

byte disp:	volume:	Content:
0	16 bit integer	Access_number
2	16 bit integer	wait_mask

9.5.8 DGDIS parameter block

Directive number= 9

byte disp:	volume:	Content:
0	16 bit integer	Access_number
2	16 bit integer	wait_mask

9.5.9 Structure of event

see general description for SMACC implementation.

9.5.10 Mask for event

see general description of DG service.

9.5.11 Error code

see general description of DG service.

A P P E N D I X A

Implementation Note

10 Implementation remarks:

10.1 What is not implemented

- Not all the functionality involved in the specification of datagram protocol is implemented. To have a light- weight implementation version, the header has got the minimum to allow rerouting inside the PS computers: CONSOLEs, FECs, SMACCs. To be precise the datagram protocol header compared with the specification gives:

Network layer protocol ID:	*dropped*
Header length:	*dropped*
Version/protocol ID:	*dropped*
Lifetime:	*dropped*
Packet type:	*dropped*
Segment length:	*implemented*
Header check sum:	*dropped*
Destination address legngth:	*dropped*
Destination address:	*implemented*
Source address length:	*dropped*
Source address:	*implemented*
Identifier:	*implemented*
Segment offset:	*implemented*
Total length:	*implemented*
Option:	*dropped*

- The implementation provides only one size of packet buffers to prevent segmenter and reassembler wasting time in reorganising packets.
- The implementation of credit is incomplete and dubious. No attempt is made to ensure that the total credit allocated to all sockets does not exceed the total buffer pool size, and a primitive over-booking technique is used in DGSND to (hopefully) avoid credit failures while sending a multipacket datagram at the same time as incoming packets are consuming credit. An error return is given back when there is not enough credit at this moment.

10.2 Data structure in DG service

The processing of the datagram through the service is based on the following data structures:

- the packet buffer control block:

packet manipulation information: fifo link
ownership information
space for TITN header
space for protocol header of level 3

space for level 4 data

- *The datagram descriptor control block:*

  datagram manipulation information: fifo link
  identifier
  fifo of packet
  age
  size
  source

- *The socket control block:*

  socket manipulation information: fifo link
  fifo of incoming datagrams
  fifo of outgoing datagrams
  current status indicator
  status of DG_set option

These packets are linked into various fifos, datagrams are always linked to the relevant socket control block, and socket control blocks are either free, or in use by a given user task.
The unused elements are linked to a fifo of free elements of the corresponding type.
All these fifo and links are set up at startup of the DG service.

10.2.1 Configuration parameter of datagram service:

The characteristic of the datagram service depend on parameters whose value are defined either at compiled time or at load time.

10.2.2 Parameters defined at compiled time:

Some parameter can be modified but need a total recompilation of the whole datagram service for all machines; these symbols are all declared in the file DG-COMN-DCL:SPLC.

- *TITNZ = 64 for the size of TITN (level 2) packet.*
- *SCAMZ = 64 for the size of SCAMAC (level 2) packet.*
  In the current implementation we must satisfy SCAMZ= TITNZ
- *STD_TIMEOUT=10, it is the lifetime in second given to a datagram*
  i.e. the time given to 'DGRCV' it, before to throw it away.

On the SMACC only:

-0:xxxx= Range of a static 16 bit integer array declared for data structure management in the file DG-SM-ZBUF-DTA:SPLC

```
INTEGER2 ARRAY ZBUF[0: xxxx]
```

10.2.2.1 Parameter defined at load time:

Configuration parameters depending on the local context, are independent of compilation. Their value is dynamically used at initialisation of the DG service:

- *On the ND100 they are defined at load time of DGservice segment.*
- *On the SMACC they are defined by a compiled module in the file DG-SERVICE-TCB:SPLC because the image builder does not accept stand unsatisfied external references.*

The value to be defined are:

- *DGSNB defines the number of socket contexts (max number of sockets opened at the same time).*
- *DGDNB defines the number of datagram headers (max number of datagrams in process at the same time).*
- *The rest of the memory is used to tailor the packet header to manipulate data of datagrams.*

On the ND100 the room dedicated for buffer of the DG service is defined in the mode file of loading:

- *xxxxxx= octal number of 16 bits words to be allocated for managing of data. Must be a multiple of page size, i.e.: 10000 (octal), in the following RT Loader command:*

  ALLOCATE-AREA DGPSG,xxxxxx

A P P E N D I X B

Reminder of basic definitions

11 Wait mask structure:

The coding of wait_mask is:

bit 0 = 1 : wait applies to all sockets opened by this user (in this case access_number is ignored).
bit 1 = 1 : timeout required
bit 2 = 1 : outside world wait required and system_mask used
bit 3 = 1 : wait till inqueue non empty (receive possible)
bit 4 = 1 : wait till outqueue empty (all sends done)
bit 5 = 1 : wait till credit >= credit request (local flow control)

Apart from bit 0, the same bits are used in return_mask to indicate the reason(s) for returning from wait.

11.1 Network address structure:

% 16 bits % ISO_code
% 16 bits % network_number
% 16 bits % station_number
% 16 bits % socket Where:

ISO_code =
bit 1 = I/G to distinguish Individual address from Group address
0 for Individual, 1 for Group.
use always 0.
bit 0 = L/U to distinguish locally from universally administered address.
use always 1.
bit 2..15 = unused bits, always= 0

network_number= to identify a physical network or subnetwork.
use the value 0.

station_number= to identify the number of the station in the network.
If the addressed station is a FEC, it is the TITN line number.
If the addressed station is a SMACC, the structure of station_number is:
8 MS bits= FEC line value
8 LS bits= SMACC number

Socket_number= has the role of NetWork Service Access Point (NSAP) selector, identifying the user in the net work

N.B.:
Since 0 is not allowed as value of a local station the SMACC numerotation should start with 1.

11.2 Allocation of socket number:

- Socket identifying services: number 1..255 (FF hexa)
- Socket identifying application: number >= 256 (100 hexa)

To access services the calling mode does not need a fixed allocation of the socket, in that case a dynamic allocation will be performed:

- *Socket dynamically allocated: negative number.*

The dynamic allocation of socket is obtained by specifying 0 as socket number in DGOPN.

11.3 Socket information

```
TYPE socket_info:
     RECORD
        local:              internet_address
        filter:             internet_address
        priority:           BOOLEAN     % 16 bits
        discard_timeout:    INTEGER     % 16 bits
        inqueue:            INTEGER     % 16 bits
        outqueue:           INTEGER     % 16 bits
        credit:             INTEGER     % 16 bits
     END RECORD
```

11.4 Error code given back by datagram service

Error codes and warning values are named as in PPL define module

```
dg_OK:                      0;
dg_queue_empty:             1;
dg_no_credit:               2;
dg_invalid_access_number:   3;
dg_invalid_credit_request:  4;
dg_invalid_timeout:         5;
dg_internal_error: 6;% Abnormal error, please contact
                        %  supervisor of running computer
dg_truncated:               7; % from DGRCV
dg_already_open:            8; % from DGOPN
dg_buffer_size_error: 9; % size not compatible with actual size
                        % of array : from DGSND
dg_not_started:            10; % DG servive not available
dg_data_empty:             11;
dg_invalid_parameter:      12;
```

Warning codes specific to dg_wait on ND100

```
-1= dg_timeout: time given for waiting elapsed,
-2= dg_prompted:  removed from time queue while waiting,
-3= dg_didnotwait: no wait done because of the bit repeat
                   which made the wait dummy.
```