GOOSY
Id.: INTRO
Version: 1.2
Date: September, 3 1987
Revised: June, 28 1988

GSI **O**nline **O**ffline **S** **Y**stem

GOOSY Manual

H.G.Essel et al.

June, 28 1988

GSI, Gesellschaft für Schwerionenforschung mbH
Postfach 11 05 52, Planckstraße 1, D-64220 Darmstadt
Tel. (0 6159) 71–0

List of Figures

Chapter 1

Preface

GOOSY Copy Right

Conventions used in this Document

$\boxed{\texttt{Fn}}$, $\boxed{\texttt{PFn}}$, $\boxed{\texttt{1}}$, $\boxed{\texttt{Do}}$, or $\boxed{\texttt{Return}}$ key — All key in frame boxes refer to the special keypads on VTx20 compatible terminals like VT220, VT320, VT330, VT340, VT420, VT520, PECAD, PERICOM terminals or DECterm windows under DECwindows/Motif on top or right to the main keyboard, to control characters, or to the delete and return keys of the main keyboard.

<Fn>, <PFn>, <KPn>, <Do>, or <Ctrl>— This is the alternative way of writing the keypad or control keys.

$\boxed{\texttt{GOLD}}$, <GOLD>— The $\boxed{\texttt{PF1}}$ key is called $\boxed{\texttt{GOLD}}$ in most utility programs using the keypad.

PERICOM— On the PERICOM terminal keyboard the function keys are marked opposite to all other terminals, i.e. the 4 $\boxed{\texttt{PFn}}$ of the rightmost VTx20 compatible keypad are named $\boxed{\texttt{Fn}}$ and the 20 $\boxed{\texttt{Fn}}$ keys on the top of each VTx20 compatible keyboard are named $\boxed{\texttt{PFn}}$ on a PERICOM.

$\boxed{\texttt{Return}}$— The $\boxed{\texttt{Return}}$ is not shown in formats and examples. Assume that you must press $\boxed{\texttt{Return}}$ after typing a command or other input to the system unless instructed otherwise.

$\boxed{\texttt{Enter}}$— If your terminal is connected to IBM, the $\boxed{\texttt{Enter}}$ key terminates all command lines.

$\boxed{\texttt{Ctrl}}$ **key** — The $\boxed{\texttt{Ctrl}}$ box followed by a letter means that you must type the letter while holding down the $\boxed{\texttt{Ctrl}}$ key (like the $\boxed{\texttt{Shift}}$ key for capital letters). Here is an example:

- $\boxed{\texttt{Ctrl}}$ `Z` means hold down the $\boxed{\texttt{Ctrl}}$ key and type the letter `Z`.

$\boxed{\texttt{PFn}}$ **key** — The $\boxed{\texttt{PFn}}$ followed by a number means that you must press the $\boxed{\texttt{PFn}}$ key and *then* type the number. Here is an example:

- $\boxed{\texttt{PF1}}$ `6` press the $\boxed{\texttt{PF1}}$ key and then type the number 6 on the main keyboard.

$\boxed{\texttt{PFn}}$ **or** $\boxed{\texttt{Fn}}$ **keys** — Any $\boxed{\texttt{PFn}}$ or $\boxed{\texttt{Fn}}$ key means that you just press this key. Here is an example:

- $\boxed{\texttt{PF2}}$ means press the $\boxed{\texttt{PF2}}$ key.

Examples— Examples in this manual show both system output (prompts, messages, and displays) and user input, which are all written in `typewriter` style. The user input is normally written in capital letters. Generally there is no case sensitive input in GOOSY, except in cases noted explicitly. In UNIX all input and with it user and file names are case sensitive, that means for TCP/IP services like Telnet, FTP, or SMTP mail one has to define node names, user names, and file names in double quotes "name" to keep the case valid for OpenVMS input. Keywords are printed with uppercase characters, parameters to be replaced by actual values with lowercase characters. The computer output might differ depending on the Alpha AXP or VAX system you are connected to, on the program version described, and on other circumstances. So do not expect identical computer output in all cases.

Registered Trademarks are not explicitly noted.

1.1 GOOSY Authors and Advisory Service

The authors of GOOSY and their main fields for advisory services are:

M. Richter GOOSY Data Management, VAX/VMS System Manager (Tel. 2394)

R. Barth GOOSY and PAW software (since 1995) (Tel. 2546)

H.G. Essel (GOOSY 1983-1993) Data Acquisition (Tel. 2491)

N. Kurz Data Acquisition (since 1992) (Tel. 2979)

W. Ott Data Acquisition (since 1994) (Tel. 2979)

People who have been involved in the development of GOOSY.

B. Dechant GOOSY software (1993-1095) (Tel. 2546)

R. S. Mayer Data Acquisition (1992-1995) (Tel. 2491)

R. Fritzsche Miscellanea (1989-1995) (Tel. 2419)

H. Grein Miscellanea (1984-1989)

T. Kroll Miscellanea, Printers (1984-1988)

R. Thomitzek Miscellanea, Printers, Terminals (1988-1989)

W. Kynast GIPSY preprocessor (1988)

W.F.J. Müller GOONET networking, Command interface (1984-1985)

H. Sohlbach J11, VME (1986-1989)

W. Spreng Display, Graphics (1984-1989)

K. Winkelmann GOOSY Data Elements, IBM (1984-1986)

1.2 Further GOOSY Manuals

The GOOSY system is described in the following manuals:

- GOOSY Introduction and Command Summary
- GOOSY Data Acquisition and Analysis
- GOOSY Data Management
- GOOSY Data Management Commands

- GOOSY Display
- GOOSY Hardware
- GOOSY DCL Procedures. GOOSY Error Recovery
- GOOSY Manual
- GOOSY Commands

Further manuals are available:

- GOOSY Buffer structures
- GOOSY PAW Server
- GOOSY LMD List Mode Data Generator
- SBS Single Branch System
- TCP-Package
- TRIGGER Bus
- VME Introduction
- OpenVMS Introduction

Part I

VAX VMS Introduction

Chapter 2

VAX Login and Logout

2.1 The Computer Account

As a new user of a GSI computer you have to be accounted for. Through this procedure the system manager (for OpenVMS see names on page **??**) sets up a disk directory for you and you will receive a username and a password.

You must change your password immediately after the very first login by the command

```
$ SET PASSWORD
```

Only after you have received an account you can proceed to any terminal for login.

2.2 General Remarks on GSI Computers

At GSI a large number of Alpha AXP or VAX computers namely Alpha workstations or VAX-stations is used by experiment groups for data collection and data analysis, for the accelerator control system, for the safety department control system, and for a printed circuit layout CAD system. The OpenVMS operating system is very popular and gratefully accepted by most experiment groups at GSI, besides the IBM mainframe under MVS and several UNIX workstations from DEC, IBM, and HP.

Most computers at GSI are connected to each other by a large Ethernet/FDDI network (FDDI Fiber Distributed Data Interface with transparent bridges to Ethernet) available in almost all rooms. This network is separated by bridges into several segments reducing the overall data traffic and avoiding disturbances of the whole network by one local error source. Several (up to 32) FDDI rings are separated by a very fast cross bar switch, the GIGAswitch, which switches FDDI packets from source rings to destination rings (different rings simultaneously). Several Alpha AXP and VAX computers are connected directly to the GIGAswitch (building their own little ring). FDDI is used mainly as a backbone for Ethernet segments. In fig. 2.1 on page 10 you see a generalized picture of the computer network.

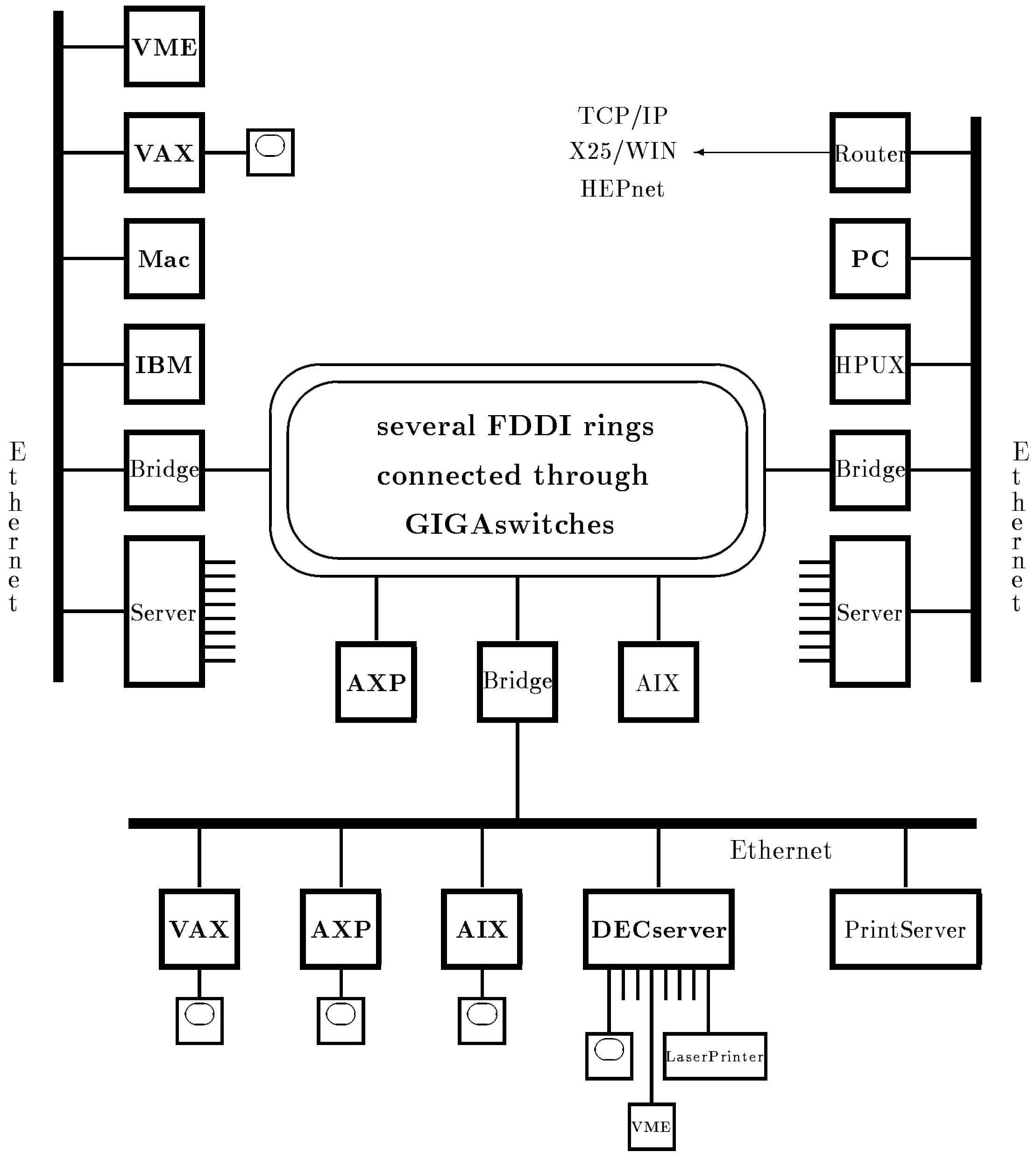


Figure 2.1: Schematic view of computers and terminals in the Ethernet/FDDI network

Although the Alpha AXP or VAXstations and the Ethernet are easy to use, users are strictly prohibited from making any changes or re-configurations on the computer hardware or on the Ethernet cabling by their own. Specifically:

- NEVER use 93 Ω IBM 327x cable or 75 Ω video cable as a thin wire Ethernet cable. Use ONLY specific 50 Ω Ethernet cables available in the GSI stock.
- Never make a stub from a T-connector to the computer, i.e. never place the T-connector close to the wall and feed only one cable to the computer, this corrupts the network.
- The T-connector of the thin wire Ethernet cabling must be as close to the computer interface as possible, i.e. from each of the two cable connectors on the wall a separate cable must go to the T-connector close to the computer.
- NEVER leave the thin wire Ethernet cable connection open. Never remove the cable bridge on the wall connectors without immediate replacement.

Everyone must contact the Computer Advisory Service (for OpenVMS, Ethernet and terminals see names on page **??**) in case of hardware re-configurations. They will provide you with the necessary service. They are also trying to keep a record of the whole network hardware which requires your help.
Keep your hands off the computer hardware and the Ethernet cabling!
All computers at GSI might be corrupted even by a single source of hardware malfunction!

Since most of the computer and network components are under maintenance contract, please, never unpack or install new computer equipment by your own (like Alpha AXP, VAXstations, DECstations, add-on memory, DEC terminals, magnetic disks, interfaces, printers, or tape drives). Call the Computer Advisory Service in any case for unpacking and installation. Otherwise GSI might run into problems with the warranty, the service and the software installation. A copy of the delivery sheet (from the stockroom personal) should be passed to the Computer Advisory Service. They also need the software license sheets (PAK) coming together with the computer. Also, never re-configure existing computer equipment by your own (like Alpha AXP, VAXstations, DECstations, add-on memory, DEC terminals, magnetic disks, interfaces, printers, or tape drives). You should also contact the Computer Advisory Service for connecting Xwindow terminals, IBM compatible PCs or Apple Macintosh to Ethernet. Not all PC Ethernet cards are supported by the network software and may disturb the network hardware.

2.3 User Interfaces to GSI Computers

The user communicates with the computers by text, graphics, or Xwindow terminals. Graphics terminals can be part of a workstation, i.e. connected directly to a computer. There are also Xwindow terminals connected via Ethernet to the computer. Text terminals and some simple graphics terminals are connected to Terminal Servers which for their part are connected to Ethernet. A Terminal Server, e.g. a DECserver, is a device connecting terminals and printers via an Ethernet coaxcable to all Alpha AXP, VAX, ULTRIX, IBM RS/6000 UNIX, HP UNIX, and OSF/1 UNIX, computers at GSI, to the IBM mainframe computer, and to several VME processor boards (OS/9, LynxOS, and pSOS). Laser printers are also connected to Terminal Servers allowing access from all computers. The DECservers are using the LAT (Local Area Transport) protocol for the connections between themselves and the computers. In other words, all computers, terminals, and printers are linked together allowing access to each other across the Ethernet/FDDI network.

There are several types of terminals available for GSI computers

1. a text (ASCII) terminal compatible to a VTx20 (e.g. VT220, VT320, VT330, VT340, VT420, VT520, PECAD, or Pericom) connected via a Terminal Server to Ethernet at GSI to login for OpenVMS, ULTRIX (on a DECstation), and MVS (on the IBM mainframe) or to VME processors. Connected to a DECserver-300 or -700 a direct TCP/IP Telnet connection can be established.
2. a text (ASCII) terminal compatible to a VTx20 connected directly to an Alpha AXP or VAX or DECstation.
3. a window graphics terminal under DECwindows/Motif connected directly to an Alpha AXP or a VAXstation or via a Xwindow terminal or an IBM PC or an Apple Macintosh remotely connected to an Alpha AXP or VAX via Ethernet.
4. a window graphics terminal connected directly to a VAXstation under the older VWS window system which will not be described in this manual.
5. a text terminal compatible VT-window (DECterm) under DECwindows/Motif (or VWS).
6. a simulation of a Terminal Server session after logged in a OpenVMS system using the DCL command `$ CON` or `$ SET HOST/LAT`
7. an IBM compatible PC with Pathworks for DOS simulating a text terminal to login on an Alpha AXP or a VAX or a DECstation.
8. an Apple Macintosh with Pathworks for Mac simulating a text terminal to login on an Alpha AXP or a VAX or a DECstation.
9. a window graphics terminal under DECwindows/Motif connected directly to a DECstation with ULTRIX or via a Xwindow terminal remotely connected to a DECstation with ULTRIX via Ethernet.
10. a window graphics terminal under AIX or HP-UX connected directly to an IBM RS/6000 or HP 9000 workstation or via a Xwindow terminal remotely connected to such a workstation via Ethernet.

11. an IBM 327x or IBM 5080 compatible terminal connected directly to the IBM mainframe.
12. remote login via TCP/IP Internet with Telnet on OpenVMS, ULTRIX, AIX, HP-UX, or MVS or any IBM PC or Apple Macintosh running TCP/IP software.
13. remote login via X-25 (DATEX-P, WIN) on OpenVMS or MVS.
14. remote login via HEPnet (DECnet) on OpenVMS.
15. remote telephone modem login on OpenVMS or MVS.
16. using TCP/IP Internet (Telnet) or HEPnet (DECnet) or X-25 (DATEX-P, WIN) to login on any computer system outside GSI running Telnet or X-25 if you have an account on such a system. You can use any OpenVMS, ULTRIX, AIX, HP-UX or MVS session, any IBM PC or Apple Macintosh running TCP/IP software to do so or you may use a text terminal connected to a DECserver-300 or -700.

Different types of terminals are handled in a different manner, some of them will be described in the following sections.

2.4 Logging In

2.4.1 The Alpha AXP or VAX Text Terminal

In this context a text terminal is any terminal out of the following types:

- all text terminals compatible to the VTx20 series (e.g. VT220, VT320, VT330, VT340, VT420, VT520, PECAD, or Pericom),
- the emulations under DECwindows/Motif (DECterm) and VWS,
- a window from a Xwindow terminal with LAT or TCP/IP,
- a connection under Pathworks for DOS and Mac,
- a remote login via Telnet (TCP/IP Internet),
- a remote login via X-25,
- a remote login via HEPnet (DECnet),
- or a remote login via telephone modems.

They are all using the same VTx20 keyboard layout (for PCs, MACs, and some Xwindow terminals with keyboard simulations). The usage of this Alpha AXP or VAX text terminal keypad (VTx20 compatible) is described in the appendix A on page 381.

The following description shows the OpenVMS login procedure for different terminal types. Normally, only the first phase of the OpenVMS login differs between the different terminal types.

1. A text terminal connected to a Terminal Server.
   After pressing `Return`, *one* of the following is possible:

   (a) The `S200xx Local>` prompt is displayed and you can proceed.

   (b) This is displayed:

   ```
   DECserver 200 Terminal Server V3.1 (BL37) - LAT V5.1
   LTA421-LTA428
   Please type HELP if you need assistance
   Enter username>
   ```

   After entering your name, you can proceed. (Note: Please, enter only your "real" name or your username.)

   (c) If your terminal has a multisession option (VT330, VT340, VT420, VT520) and the terminal and the Terminal Server port is enabled to use this option there will be the following highlighted prompt in the lowest line on your terminal

   ```
   Service Name:
   ```

   You may now enter a desired service (see later) or just `Return` which will produce the prompt

   ```
   S200xx Local>
   ```

   At this point you may enter

   ```
   S200xx Local> HELP
   ```

   showing the available commands. Of the possible commands only one is of importance at this time, namely:

   ```
   S200xx Local> SHOW SERVICES
   ```

   This shows the services available. In the following example only some of the offered services are shown:

   ```
   Service Name    Status      Identification
   ...
   AXP601         Available   A X P 6 0 1   Digital 2100 Server Model A500MP
   AXP602         Available   A X P 6 0 2   Digital 2100 Server Model A500MP
   AXP610         Available   A X P 6 1 0   DEC 3000 Model 400
   AXP611         Available   A X P 6 1 1   DEC 3000 Model 300
   ```

```
...
CLEX1         Available    The GSI Online Cluster 1
...
DSAA          Available    ULTRIX 4.4 (RISC)
DSAB          Available    ULTRIX 4.4 (RISC)
...
GOOSY         Available    The GSI Online Cluster 1
...
IBM           Available    IBM-DECserver Terminal Lines
...
MVIIC         Available    O T T O       VAXstation II
MVIID         Available    P A U L A
MVIIE         Available    M V I I D     MicroVAX II
MVIIG         Available    M V I I G     VAXstation 3600 Series
...
V6000A        Available    V 6 0 0 0 A   VAX 6000-340
...
VME1          Available    VME El/Ex-Lab S200CC Port 1
VME2          Available    VME El/Ex-Lab S200CC Port 2
VME3          Available    VME El/Ex-Lab S200CC Port 3
...
VSAA          Available    W S A A       VAXstation 2000
VSAB          Available    W S A B       VAXstation 3100/GPX
VSAC          Available    W S A C       VAXstation 3100/GPX
...
VSCG          Available    W S C G       VAXstation 4000-60
VSCH          Available    W S C H       VAXstation 4000-60

S200xx Local>
```

If you enter the GOOSY service, you will be connected to a GSI Online/Offline System VAX, i.e. either V6000A (FRITZ) or VSCN, whichever has more CPU time available.

If your have been logged in already on a OpenVMS system you may simulate such a Terminal Server command by using the command

```
$ SSERVICE
```

to get a list of all services available.

You can enter an available (Status: `Available`) service by typing:

```
S200xx Local> CONNECT service
or just
S200xx Local> C service
```

or from a current OpenVMS session such a Terminal Server connection can be simulated by using the command

```
$ CON service
or
$ SET HOST/LAT service
```

`service` should be replaced by the available service name. For example: To enter AXP601, the ALPHA AXP node AXP601, you would type:

```
S200xx Local> CONNECT AXP601
or just
S200xx Local> C AXP601
```

Now your text terminal is connected to the desired Alpha AXP. Proceed to login.

2. If your text terminal is connected directly by hardware to an Alpha AXP or a VAX computer at GSI proceed to login.

3. If you are using a PC with MicroSoft Windows start the eXcursion Control Panel and from that the Applications icon, and from that the OpenVMS Terminal application, the Host name (e.g. V6000A) and then click Run. You will be prompted for the OpenVMS account user name and its password. Please, type in both in capital letters and go from the Username input field to the Password field not by the `Return` key but by the `Tab` key. or by a mouse click to the Password field. Otherwise your OpenVMS account might be disabled (if so, please contact the OpenVMS Advisory Service). Then proceed to login by clicking the OK field. For configuration help contact the OpenVMS Advisory Service (for OpenVMS see names on page **??**).

4. If you are using a text terminal simulated on a Apple Macintosh the network components must have been installed and configured. For help contact the Advisory Service (for OpenVMS see names on page **??**). Select the menus Apple → MacTerminal Folder → MacTerminal. Set your default terminal and communication setups with the menus Setting → Terminal... or Setting → Connections... → LAT Tool or → CTERM Tool (DECnet). Then open the terminal using the menu Session → Open Connection. Then proceed to login.

5. If you want to login from another OpenVMS session using DECnet within the GSI use the command

```
$ SET HOST node
or just
$ HOST node
e.g.
$ HOST AXP601
```

and proceed to login.

6. If you login from remote via Telnet (TCP/IP Internet) you need the Internet address or name of the GSI computer you would like to connect to. The GSI naming convention is: `node.gsi.de`, e.g. `axp601.gsi.de`, `v6000a.gsi.de` or `mvs.gsi.de` for the IBM mainframe. Not all computers at GSI are reachable from remote. You can get the valid Internet addresses using the DCL command `$ UCX SHOW HOST name`. The valid Internet address should be typed out. To get detailed information and the allowance for remote login please contact the Advisory Service (for OpenVMS see names on page **??**). You connect to a remote host via Telnet with the DCL command

   ```
   $ TELNET name
   or
   $ TELNET
   TELNET>OPEN name
   or
   TELNET>CONNECT name
   e.g.
   $ TELNET "vscn.gsi.de"
   or
   $ TELNET
   TELNET>OPEN "vscn.gsi.de"
   ```

7. If you login from remote via X-25 (DATEX-P, WIN) you need the X-25 address of the GSI computer you would like to connect to. Only the VSCN and the IBM mainframe are reachable from remote. To get detailed information and the allowance for remote login please contact the Advisory Service (for OpenVMS see names on page **??**).

8. If you login from remote via a telephone modem you need the telephone number of the GSI computer modem and your own telephone number must be registered for the automatic call back facility. To get detailed information and the allowance for remote login please contact the Advisory Service (for OpenVMS see names on page **??**).

9. If you login from remote via HEPnet (DECnet) you need the DECnet address or name of the Alpha AXP or VAX computer you would like to connect to. Not all computers at GSI are reachable from remote. To get detailed information and the allowance for remote login please contact the OpenVMS Advisory Service (see names on page **??**).

Now you can login on the Alpha AXP or VAX system by entering your username.

```
AXP601 (Digital 2100 Server Model A500MP, OpenVMS-V6.1) Please login
```

```
Username: user
```

Type in the username you received from the OpenVMS system manager when you got your account

followed by the password belonging to your username

```
Password: password

   Welcome to OpenVMS AXP (TM) Operating System, Version V6.1 on node AXP601
    Last interactive login on Wednesday, 10-AUG-1994 20:38:43.48
    Last non-interactive login on Friday, 30-AUG-1991 17:25:28.40

 System up since 26-JUL-1994 08:22:21.00

 Hello <user> is AXP601, have a nice day !
 You logged in at 18:43:19 on Thursday, August 11, 1994
 You are on terminal AXP601$RTA5.

There are no news for GSI
News for the following facilities are available:
CERNLIB
GOOSY
GSI
UPDATE
AXP601 $
```

You are now connected, and the DCL (Digital Communication Language) Prompt (`AXP601 $`) is displayed. That means, that you are presently on the DCL Command Level. In this manual the prefix defining the Alpha AXP or VAX you are logged in (like `AXP601 $` in the above example) will be neglected, i.e. only `$` is used in this manual for the DCL prompt. You may change the DCL prompt to any string you like by the DCL command `$ SET PROMPT=string`. The terminal name differs depending on the way you logged in (LTAxxxx for LAT, TNxxxx for Telnet, and so forth). Now proceed to 'First Steps under OpenVMS' on page 21.

2.4.2 The Xwindow Terminal

In this context a Xwindow terminal is any terminal out of the following types:

- DEC VXT-2000 terminal with a DEC VT220 like keyboard

- Tektronix TekXpress terminal with a DEC VT220 like keyboard
- Tektronix TekXpress terminal with an IBM-PC like keyboard

In the case of a DEC VXT-20000 you may connect to a host as a TCP/IP Xwindow, a LAT Xwindow, a TCP/IP terminal, or a LAT terminal session. In case of a Xwindow session to a VAX or Alpha AXP the Motif session manager and the Motif windows manager will be started automatically.

In case of a Tektronix TekXpress terminal you may connect to a VAX or Alpha AXP as a TCP/IP terminal (e.g. `OPEN AXP601`) or a LAT terminal session. In these cases you must start the Motif session manager and the Motif windows manager explicitly with the DCL command

```
$ XSESSION name
e.g.
$ XSESSION XWTAD
or
$ XSESSION 140.181.96.13
```

where 'name' stands for the terminal your are working on. It might be its name or the Internet address. The command XSESSION will connect to this terminal for opening the display windows.

Only in case of a Tektronix TekXpress terminal with an IBM-PC like keyboard not all keys are mapped corresponding to a DEC VT220 like keyboard. This makes problems mainly for the editors. Therefore specific calls for the two main editors are available

```
$ XEDT file
or
$ XLSE file
```

With these specifically initialized editors the non-functional keypad keys $\boxed{\texttt{PF1}}$ to $\boxed{\texttt{PF4}}$ are simulated by the keys $\boxed{\texttt{F9}}$ to $\boxed{\texttt{F12}}$ of the IBM-PC like keyboard.

2.4.3 The DECwindows/Motif Terminal

A DECwindows/Motif session can be started on each Alpha AXP or VAXstation graphics terminal connected directly to an Alpha AXP or a VAXstation, from a Xwindow terminal connected via Ethernet or from a Xwindow terminal emulation on a PC or Mac running Pathworks. Only for directly connected graphics terminals the login prompt with the Digital logo is seen on your screen. In all other cases you must connect your Xwindow terminal or the emulation to an Alpha AXP or a VAX getting this logo after the successful connection. If you login via `$ SET HOST 0` (i.e. on the same DECnet node) no DECwindows application can be started directly.

Type in your username and your password each followed by a `Return`.

The following is a brief description of the handling and the set-up of the window system DECwindows/Motif:

When you have more than one window open, you must make the window active you want to work with. To do so, point to a location in the window or window frame by moving the mouse and click the left mouse button (MB1). The window moves to the front of the screen and the window frame is highlighted.

If one window partially obscures another, you might want to arrange them so that each is visible. To move a window, position the pointer anywhere in the window's title bar (except on a button in this field), press and hold the left mouse button (MB1), and drag the window outline to the new location.

You can change the size of your windows to suit your needs by using the window's resize borders. To change the size of a window, position the pointer on one of the window's resize borders. The pointer changes into a resize cursor. Press and hold MB1 and drag the resize cursor to the size you want.

If you have several applications running at the same time, you can free up space on your screen by minimizing a window (shrink to an icon). All processes continue to execute while the application window is an icon. To minimize a window, point to the window's minimize button (the left button on the upper right corner of the window) and click MB1. To restore an icon to a window, point to the icon and double click MB1.

The window menu contains menu items for working with windows. To display the window menu, click on the Window Menu button (button on upper left side of a window).

After starting a session, you use the Session Manager to manage your session and your workstation environment. When you start a session, DECwindows/Motif displays the Session Manager's menu bar. You can use the Session Manager's Options menu to customize your environment. Select the Options field with the pointer and click MB1. You will see a menu of options including

Automatic Startup...
(to select applications starting automatically during login to Motif.
The Window Manager <u>must</u> be selected in the Automatic Startup.),
Window... (window layout),
Menus... (selection of applications available in the menu),
Menu Bar... (layout of the Session Manager Menu Bar),
Pause Screen...,
End Session Prompts...,

Screen Background... and Window Colors...,
Keyboard.. (with keyclick setup in it),
Language... (select English or German e.g. for DECwrite or DECdecision),
Pointer...,
Security... (to allow other users the opening of windows on your screen).

You can select any option with the pointer and a click with MB1. After setting your environment you can save these settings with the option 'Save Session Manager'.

To put your current session on hold, choose 'Pause' from the Session Manager's Session menu. To end your session, choose 'End Session' from the Session Manager's Session menu.

Standard applications include DECterm, Bookreader, Calendar, Calculator, Mail, Paint, or FileView. Short descriptions are given in appendix **??** on page **??**. Details can be found in Help or with the Bookreader (see the chapter **??** on page **??**).

The DECterm option simulates a VT330 (black/white) or a VT340 (color) terminal. Each selected DECterm opens a window on the screen and runs the user's login procedure automatically without asking again for a username or password.

2.4.4 First Steps under OpenVMS

You must change your password immediately after the very first login by the command:

```
$ SET PASSWORD
```

If you are using a PECAD terminal the first time give the command

```
$ PEVAX
```

to set up the PECAD terminal correctly. With the command this setup is store permanently.

Now you can get information about the directory of your private files by entering

```
$ DIRECTORY /DATE/SIZE=ALL
or just
$ DIR
```

Your directory is displayed as e.g.:

```
Directory GSI$ROOT:[user]

LOGIN.COM;1              2/3       11-JUN-1992 18:25

Total of 1 files, 2/3 blocks.
```

The directory displays the following information:

- The name of files; in the above example it is `LOGIN`
- The type extension of files; above it is `COM`
- The version number of files; above it is `1`. The version number is automatically upgraded each time you save the file with the same name, e.g. when you edit an existing file the new, changed file will get a new, higher version number.

`LOGIN.COM` is, as its name suggests, a command procedure executed whenever you log in. You can add commands to this file. These commands are then executed whenever you log in. An example of a login file is shown in appendix B on page 385. You may get this template login file by the DCL command:

```
$ COPY/LOG GOO$EXE:USER_LOGIN.COM SYS$LOGIN:*
```

Do not forget to edit this template for your personal needs.

2.4.5 The IBM-Terminal (Ethernet) Connection

Terminals hooked to Terminal Servers on the Ethernet network or any active OpenVMS session can access the IBM via LAT.

A terminal connected to a Terminal Server you want to use with the IBM must respond after pressing $\boxed{\texttt{Return}}$ in one of the following ways:

1. The `S200xx Local>` prompt is displayed and you can proceed.
2. This is displayed:

   ```
   DECserver 200 Terminal Server V3.1 (BL37) - LAT V5.1
   LTA421-LTA428
   Please type HELP if you need assistance
   Enter username>
   ```

   (Note: Please, only enter your "real" name or your username.) After entering your name the prompt `S200xx Local>` will appear and you can proceed.

When you are in the `S200xx Local>` mode of a Terminal Server,

```
S200xx Local> HELP
```

will show the available commands. Of the possible commands only one is of importance at this time, namely:

```
S200xx Local> CONNECT IBM
```

If you want to connect from a running OpenVMS session give the DCL command

```
$ SET HOST/LAT IBM
or just
$ CON IBM
```

Now your terminal or session is connected to the IBM.

After your are connected to the IBM first type `Ctrl` `G`, the Master Reset of an IBM terminal line to cleanup the communication line to the IBM. Now the GSI logo with the IBM terminal device number VDnn will appear on the screen. You can login on the IBM system by typing in your account string followed by the `Enter` key and then after the prompt your password followed by the `Enter` key.

The Enter key is the command line delimiter on the IBM and **not** the Return key which just will move the cursor downwards.

After connecting to the IBM mainframe the following commands and keys are available (see also the keypad layout in figure 2.2 on page 25):

`Ctrl` `G` — Master Reset; should be used directly after `CONNECT IBM`.

`Enter` — The `Enter` key is the command line delimiter on the IBM and **not** the `Return` key which just will move the cursor downwards.

`Insert Here` — The `Insert Here` key switches from the default overstrike mode to the insert mode and back.

`PF4` — The `PF4` key is the **Attention** key for the IBM.

`Ctrl` `R` **or** `Ctrl` `G` — Error Reset; this should be used if the cursor hangs.

`Ctrl` `X` — Flush the input buffer

`Ctrl` `V` — Reshow the last logical screen

All terminals connected to the IBM are initialized for VT220 operation and also provide PERICOM or PECAD graphics, if wanted (not available for OpenVMS sessions connected to the IBM).

From an Alpha AXP or VAX you may also connect to the IBM MVS mainframe with the TN3270 utility which connects your VT220 like terminal via TCP/IP Telnet. The corresponding DCL command is

```
$ TN3270 mvs
```

You will get the IBM Netview Access Services panel for logging in. Please ask the Computer Center operators (room 1.250, tel. 2515) for a specific Netview Access Services account. Type in your username and password. Remember: the $\boxed{\text{Enter}}$ key is the command line delimiter on the IBM and **not** the $\boxed{\text{Return}}$ key which just will move the cursor downwards. After entering your correct username and password you will get to the Application Selection input panel. After selecting `TSOPASS` you will be ask `ENTER CURRENT PASSWORD FOR username-`. Enter your password again to get finally logged in to a MVS/TSO session. After `LOGOFF` your terminal input will return to the Alpha AXP or VAX again.

Device Separation is supported with another graphic terminal connected to a Terminal Server. That means the separation of an alphanumeric terminal for commands and a graphic terminal on the IBM. (Notice: An IBM terminal connected from a OpenVMS session has no graphics options available!) You have to connect the graphics terminal with the same procedure as your alphanumeric terminal to the IBM. By typing $\boxed{\texttt{Ctrl}}$ `G` on the graphics terminal after the connection you will get the IBM VDnn device number of that terminal. Do not log in on the graphics terminal to the IBM, because you want to use it with the device separation as an output device only.

Please, do not forget to free your port after you logged off from IBM (as the GSI logo with the IBM terminal device number VDnn appears). Otherwise the network still holds the connection and makes the port unavailable to other users. If your are on a terminal connected directly to a Terminal Server do this by pressing $\boxed{\texttt{F5}}$ (or $\boxed{\texttt{Ctrl}}$ $\boxed{\texttt{F5}}$ and $\boxed{\texttt{Return}}$ on a PECAD) to enter Terminal Server "local" mode. The `S200xx Local>` prompt should appear. Then enter

```
S200xx Local> DISCONNECT SESSION n
or just
S200xx Local> DIS SESSION n
```

`n` should be replaced with the session number used for the IBM. To check which one it is, you can enter `SHOW SESSIONS`.

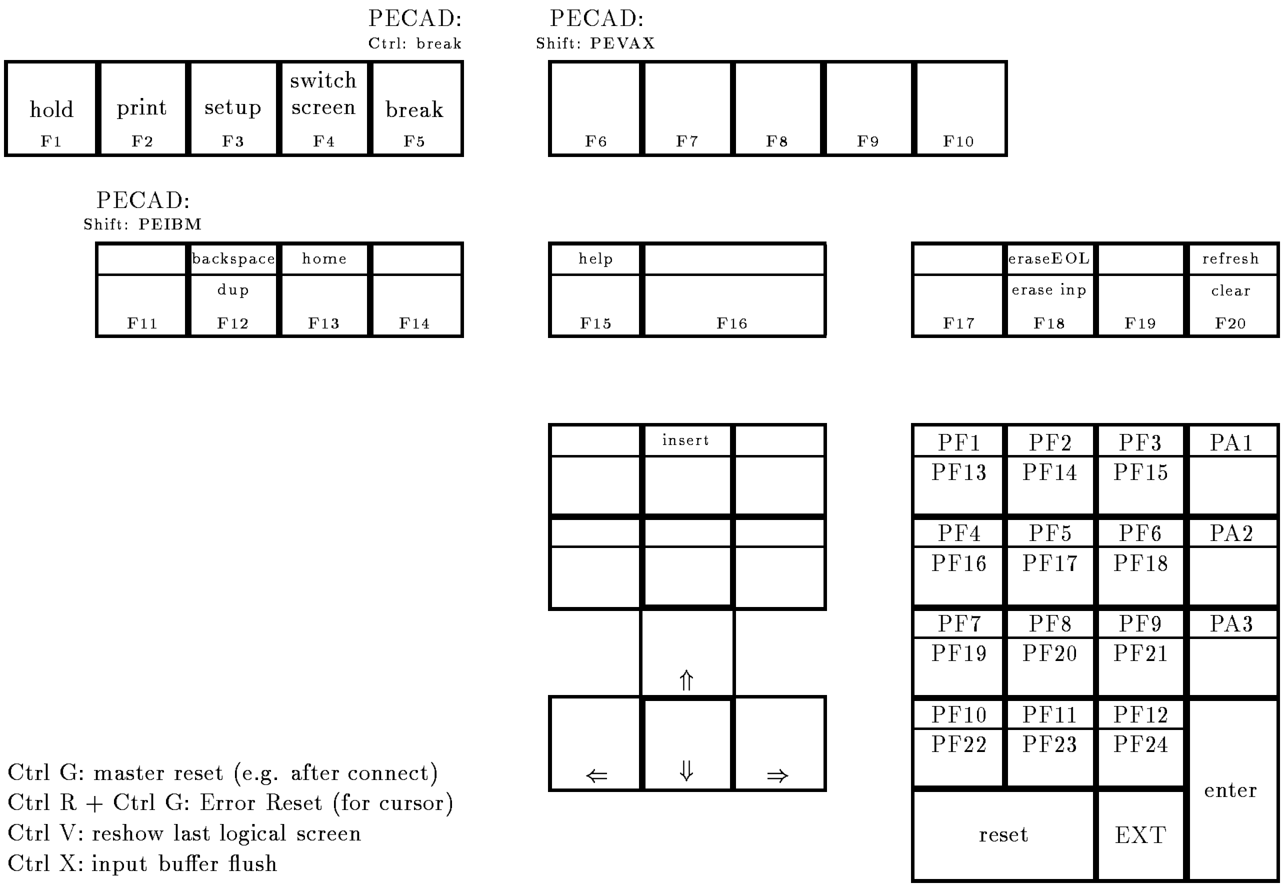
If you use a PECAD terminal on Alpha AXP or VAX and IBM alternately, use the DCL commands `$ PEVAX` or `$ PEIBM` respectively when you are logged in to an Alpha AXP or a VAX to set the terminal characteristics in the right way.

If you have been connected from a running OpenVMS session disconnect from the IBM by the key $\boxed{\texttt{Ctrl}}$ $\boxed{\backslash}$. You will be back to your original OpenVMS session.

In the figure 2.2 on page 25 you see the IBM keypad layout.

2.5 Logging Out From the Alpha AXP or VAX

Logging out from an Alpha AXP or a VAX is as easy as logging in, you simply enter the following:



Ctrl G: master reset (e.g. after connect)
Ctrl R + Ctrl G: Error Reset (for cursor)
Ctrl V: reshow last logical screen
Ctrl X: input buffer flush

The upper key values are the simple key hits, the lower are entered with a preceding EXT-key hit

Figure 2.2: The Special Keypad Layout for IBM.

```
$ LO
```

You will be asked

```
$ PURGE SYS$LOGIN:[...]*.* ? (Y,N def: N)
```

If your answer is yes (`Y`), all the old versions of all your files are deleted, and only the highest versions are kept. This is very useful, since old versions are usually not needed anymore. If, however, you answer no (`N`) or just press `Return` the old versions will be kept. After the answer the computer displays the following:

```
 Goodbye <user>, you are leaving Digital 2100 Server Model A500MP AXP601
 Have a nice time
  <user>       logged out at 11-AUG-1994 19:04:45.17
```

After logging out one is again returned to the Terminal Server "local" mode or the original OpenVMS session depending on the login method. On a Terminal Server session, the prompt `S200xx Local>` appears. The `LOGOUT` command at this level logs you out of the Terminal Server and also terminates all remaining sessions.

In case of a DECwindows/Motif session you may logout each DECterm window individually or you may exit the whole session. Therefore, select within the Session Manager Session menu the End Session option. You will be asked whether really to leave the whole DECwindows session or not. If you select YES, all windows are closed and the login window with the Digital logo appears on the screen. If you have used DECwindows/Motif from a DEC Xwindow terminal VXT-2000 the session termination will automatically reboot the whole terminal. This behavior seems to be obscure but it is correct.

2.6 Terminal Server Sessions

While using a terminal connected to a Terminal Server, you can have different sessions of the Terminal Server. You can have only two sessions if you use the multisession option together with a VT330, VT340, VT420, or VT520 terminal.

Once you have started the first session (by logging in), you can "break" out of your session by pressing $\boxed{\texttt{F5}}$ (or $\boxed{\texttt{Ctrl}}$ $\boxed{\texttt{F5}}$ and $\boxed{\texttt{Return}}$ on a PECAD). After the `S200xx Local>` prompt connect to any Alpha AXP or VAX or to the IBM by typing **`CONNECT service`** and then simply log in to the chosen service. The number of simultaneous sessions is limited by default to 4. You can have only two sessions if you use the multisession option together with a VT330, VT340, VT420, or VT520 terminal.

You can move through your established sessions by pressing $\boxed{\texttt{Ctrl}}$ \, or by breaking out to the Terminal Server local mode with key $\boxed{\texttt{F5}}$ (or $\boxed{\texttt{Ctrl}}$ $\boxed{\texttt{F5}}$ and $\boxed{\texttt{Return}}$ on a PECAD) and then using the following `S200xx Local>` commands: `FORWARD` or `BACKWARD`.

If you are using the Multisession option together with a VT330, VT340, VT420, or VT520 terminal the `FORWARD` and `BACKWARD` switches are not allowed. You switch between the two possible session using the $\boxed{\texttt{F4}}$ key of your keyboard. You may also split and unsplit the screen by using $\boxed{\texttt{Ctrl}}$ $\boxed{\texttt{F4}}$.

Chapter 3

Getting Interactive Help on VAX

3.1 OpenVMS DCL HELP

To obtain on-line documentation for a command, enter the command HELP with the name of the command as a parameter.

```
$ HELP ALLOCATE

ALLOCATE

     Provides your process with exclusive access to a device until
     you deallocate the device or terminate your process. Optionally
     associates a logical name with the device.

     Requires read (R), write (W), or control access.

     Format

       ALLOCATE  device-name[:][,...] [logical-name[:]]




  Additional information available:

  Parameters Qualifiers
  /GENERIC   /LOG
  Examples

ALLOCATE Subtopic?
```

If you need help, but do not know what command or system topic to specify, enter the command `HELP` with the word `HINTS` as a parameter. Each task name is listed in the `HINTS` text is associated with a list of related command names and system information topics.

```
$ HELP HINTS

HINTS

   Type the name of one of the categories listed below to obtain a list
   of related commands and topics.  To obtain detailed information on a
   topic, press the RETURN key until you reach the "Topic?" prompt and then
   type the name of the topic.

   Topics that appear in all upper case are DCL commands.

  Additional information available:

  Batch_and_print_jobs  Command_procedures    Contacting_people
  Creating_processes    Developing_programs   Executing_programs
  Files_and_directories Logical_names         Operators_in_expressions
  Physical_devices      Security              System_management
  Terminal_environment  User_environment

HINTS Subtopic?
```

When HELP prompts you for a topic or subtopic, you can enter one of the listed subtopics to obtain additional information (command and topic names can be abbreviated). Alternatively, you can press Return to move back a level, enter a question mark to redisplay the current text, or press Ctrl Z to exit.

Using wildcard characters when specifying a topic allows you to obtain various amounts of information.

`HELP command` — The command or topic and all related information.

`HELP command *` — All related information on that HELP level.

`HELP com*` — All commands or topics beginning with the specified character(s).

`HELP *` — All the commands and topics available in the HELP file.

`?` — Get the last seen help information on each HELP level

To get help information from the various HELP libraries listed at the end of the first HELP level type the following, for example:

```
$ HELP @GSIHELP GSILOG
```

The libraries of general interest are:

`GSIHELP:`	Several general utilities available at GSI, i.e. TEX
`SYSMSGHELP:`	OpenVMS system error message descriptions
`KERMITSYS:`	KERMIT help library
`VPW:`	FermiLab software like DTC and CALC
`UTILITY:`	GSI and GOOSY command procedures
`COMMAND:`	GOOSY commands
`PROGRAM:`	GOOSY main programs
`MODULE:`	GOOSY program modules
`MESSAGE:`	GOOSY error message descriptions
`RECOVER:`	GOOSY error recovery descriptions
`DMTYPES:`	GOOSY data element declarations

3.2 DECwindows/Motif Bookreader

If your are working directly on an Alpha AXP or a VAXstation or via a Xwindow terminal on an Alpha AXP or a VAXstation running DECwindows/Motif, and only then, a general utility is available to get all OpenVMS manuals on-line in windows on your screen.

The manuals are stored on several CD-disks mounted on a centralized DEC InfoServer connected to Ethernet only. This allows read access from all Alpha AXP and VAXstations at GSI to the same CD-disk drives.

Select from the Session Manager window Applications menu the Bookreader item. It will pop up a directory of available libraries. Select the Online Documentation Library Contents with a mouse MB1 (left mouse key) double click in that line. The sub directory appears. Select the Master Listing line with a mouse MB1 double click. The directory of all main documents appears in alphabetical order. Select the topic you are interested in, e.g. DEC FORTRAN, with a mouse MB1 double click on this line. The list of all DEC FORTRAN manuals appears from which you may select the desired one, e.g. DEC FORTRAN Language Reference Manual. Select this line with a mouse MB1 double click. A new window pops up with the Contents of this manual. Select the topic within this contents with a mouse MB1 double click. A new window pops up showing the start page of this topic. You can navigate through this topic or the whole manual using the Screen or Topic arrows on the bottom line.

Selecting in the View menu the Hotspots option by pressing and holding MB1 then moving the mouse cursor to the this line and release MB1 on the line Hotspots. Hotspots are cross references in the manual text embraced by a frame box, e.g. $\boxed{\text{Figure 3-1}}$. Select such a Hotspot with a mouse MB1 double click. A new window pops up showing the corresponding information, e.g. the Figure 3-1.

Close the text and Hotspot windows by selecting the Close option on the bottom line of the windows.

In the manual's Contents window you may select from the View menu the list of the contents (start default), examples, figures, tables, or the index. The contents can be collapsed or expanded.

Leave the manual by selecting from the File menu the Close Book option. Exit the Bookreader by selecting from the File menu the Exit option or type $\boxed{\texttt{Ctrl}}$ `e` (the keyboard $\boxed{\texttt{Lock}}$ to upper case letters should not by active).

3.3 CNEWS

News of general interest are stored by the computer Advisory Service in a specific facility called CNEWS. This facility is available on all computer platforms at GSI as OpenVMS, AIX, HP-UX, and LynxOS. The news are ordered by the field of interest into the following topics: CERNLIB, DAQ, GOOSY, and GSI. Each article is indexed within each topic. For new, unread articles you will get informed briefly during the login procedure.

You will get a list of available topics by the command

```
$ CNEWS topic
```

You select the unread news of a specific topic by the DCL command

```
$ CNEWS topic
e.g.
$ CNEWS GSI
```

A list of unread news are written to the screen.

You can get a short overview about all news including the unseen by the DCL command

```
$ CNEWS topic /ALL    or    $ CNEWS topic -a
e.g.
$ CNEWS GSI /ALL      or    $ CNEWS GSI -a
```

You can read a specific news article selecting its index

```
$ CNEWS topic n
E.G.
$ CNEWS GSI 3
```

To mark all news indices of a specific topic as 'seen' type

```
$ CNEWS topic /SEEN   or    $ CNEWS topic -s
```

A menu is invoked for this command by typing

```
$ CNEWS ?
```

3.4 WorldWideWeb WWW

The WorldWideWeb (WWW or W3) is the universe of network-accessible information, an embodiment of human knowledge. It is an initiative started at CERN , now with many participants. It has a body of software, and a set of protocols and conventions. W3 uses hypertext and multimedia techniques to make the web easy for anyone to roam, browse, and contribute to. Future evolution of W3 is coordinated by the W3 Organization. The World Wide Web is the vision of programs that can understand the numerous different information-retrieval protocols (FTP, Telnet, NNTP, WAIS, gopher, ...) in use on the Internet today as well as the data formats of those protocols (ASCII, GIF, PostScript, DVI, TeXinfo, ...) and provide a single consistent user-interface to them all. In addition, these programs would understand a new protocol (HTTP) and a new data format (HTML) both geared toward hypermedia.

Documents on the Web are referred to using URLs (Uniform Resource Locators). An URL looks like http://www.vuw.ac.nz/campus/home.html. It consists of three parts the method of retrieving the document (http), an option machine name (www.vuw.ac.nz) and a pathname (/campus/home.html). The URL format is nearly an Internet standard. Think of the so called Uniform Resource Locator (URL) as a networked extension of the standard filename concept: not only can you point to a file in a directory, but that file and that directory can exist on any machine on the network, can be served via any of several different methods, and might not even be something as simple as a file: URLs can also point to queries, documents stored deep within databases, the results of a `finger` or `archie` command, or whatever.

The GSI home page is accessible via the URL "http://www.gsi.de/gsi.html". You will find a lot of information like this manual in the GSI WWW pages.

To start WWW on OpenVMS you must call `PUBLICLOGIN` first, e.g. within your `LOGIN.COM` procedure (see or copy example `GOO$EXE:USER_LOGIN.COM`). Call the WWW on OpenVMS if you are running a Xwindow (Motif) session on a workstation or a Xwindow terminal by typing

```
$ XWWW.
```

3.5 Interactive Training Courses

There is a general interactive training course available for VAX OpenVMS users on a text terminal or compatible DECterm window. Enter the interactive training course from any running OpenVMS session (on a VAX from the CI cluster, only, not on Alpha AXP) using the DCL command

```
$ STUDENT
```

You will be guided through the course by a menu. Please, follow the instructions strictly, i.e. read all comments presented by the guide.

Chapter 4

Program Development on VAX

4.1 Editing

We recommend to use the language sensitive editor `LSEDIT` invoked from a text terminal by

```
$ LSEDIT filename.type    or just   $ LSE filename.type
!If you edit the same file again just type
$ LSE
!The editor remembers the last filename edited.
```

You may also start LSEDIT from a DECwindows/Motif DECterm with

```
$ LSEDIT/INTERFACE=DECWINDOWS filename.type
or just
$ LSE/INT=DECW filename.type
```

or select from the DECwindows/Motif Session Manager Application Menu LSEDIT. A new window pops up for LSEDIT. If this entry is not part of the Application Menu select first the option from the Session Manager Option menu. A new window Menus pops up. Select Applications from the Menu Names by a single MB1 mouse click, write `LSEDIT` in the field just above DCL Command and write `@VUE$LIBRARY:LSE$EDIT.COM` in the field right to DCL Command. Select the up-arrow right from Optional Qualifiers which places LSEDIT in the Item Names list. Select LSEDIT in this list and select the left-arrow between the Item Names list and the Applications list which places LSEDIT finally into this list. Exit by selecting the OK button. From the Session Manager window select within the Options menu the Save Session Manager option to save the current set-up for later login.

In the following we summarize the main `LSEDIT` editor features: (In the LSEDIT manual and the LSEDIT help the $\boxed{\texttt{PF1}}$ key is called $\boxed{\texttt{GOLD}}$ key.)

- Files have never line numbers or NULLs like on the IBM. Nevertheless you can get the line number where the cursor is located currently by typing $\boxed{\texttt{GOLD}}$ $\boxed{\texttt{Prev Screen}}$. You can also move the cursor to a desired line number n by typing $\boxed{\texttt{GOLD}}$ $\boxed{\texttt{Insert Here}}$

- Upper and lower case characters are always displayed as they are.
- You switch between `insert` and `overstrike` mode by $\boxed{\texttt{Ctrl}}$ `A`.
- A new line is entered by the $\boxed{\texttt{Return}}$ key instead of the $\boxed{\texttt{Enter}}$ key. To proceed the cursor to a new line use $\boxed{\texttt{0}}$ or $\boxed{\Downarrow}$ for going down or $\boxed{\Uparrow}$ for going up with the cursor within your text.
- The screen splitting is entered or left by $\boxed{\texttt{PF1}}$ `=`. To switch between the two windows, use $\boxed{\texttt{F20}}$ or $\boxed{\texttt{GOLD}}$ $\boxed{\Uparrow}$.
- To delete the character left from the cursor, use $\boxed{\texttt{Delete}}$. To delete the character on the cursor position (right), use $\boxed{\texttt{,}}$. You can undelete the character by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{,}}$.
- To delete a word left from the cursor, use $\boxed{\texttt{F13}}$. To delete a word on cursor position (right), use $\boxed{\texttt{-}}$. You can undelete the word by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{-}}$.
- Delete a line from the cursor position left by $\boxed{\texttt{Ctrl}}$ `U`. To delete the rest of a line up to the begin of the next line right from the cursor position use $\boxed{\texttt{PF4}}$. To delete the rest of a line right from the cursor position use $\boxed{\texttt{PF1}}$$\boxed{\texttt{2}}$. You can undelete the line by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{PF4}}$.
- Search a string by $\boxed{\texttt{Find}}$ or $\boxed{\texttt{PF1}}$ $\boxed{\texttt{PF3}}$. The string to be searched is prompted. Upper and lowercase characters are equivalent. Note that the search with wildcards is not supported by default! To search with the wildcards * or % use $\boxed{\texttt{Do}}$ `SEARCH/PATTERN` $\boxed{\texttt{Return}}$ and then enter the string containing the wildcards. If you want to include the characters * or % in the string you search precede them by a \, e.g. `adam*\%` would search for 'adam' followed by any number of characters followed by one % character.
  Pressing just $\boxed{\texttt{PF3}}$ continues the search in any case. The search direction is defined by $\boxed{\texttt{4}}$ for downwards and $\boxed{\texttt{5}}$ for upwards. The search string is replaced by the content of the paste buffer (see below) by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{9}}$. Search and replacement (substitution) can be done at once by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{Enter}}$.
- Shift text left use $\boxed{\texttt{F11}}$, shift text right use $\boxed{\texttt{F12}}$.
- Move the cursor to the end of a line by $\boxed{\texttt{2}}$.
- Move the cursor to start of the next line by $\boxed{\texttt{0}}$.
- Scroll text several lines up/down by $\boxed{\texttt{Next Screen}}$ or $\boxed{\texttt{Prev Screen}}$ or $\boxed{\texttt{8}}$. The direction for $\boxed{\texttt{8}}$ is defined by $\boxed{\texttt{4}}$ for downwards and $\boxed{\texttt{5}}$ for upwards.
- The direction of move cursor, scroll text, and search commands is changed by $\boxed{\texttt{4}}$ for downwards and $\boxed{\texttt{5}}$ for upwards. The direction will be kept until $\boxed{\texttt{4}}$ or $\boxed{\texttt{5}}$ are pressed again.
- Move the cursor to the bottom of the text buffer by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{4}}$.

- Move the cursor to the top of the text buffer by $\boxed{\texttt{PF1}}$ $\boxed{\texttt{5}}$.
- Rewrite (refresh) the whole screen by $\boxed{\texttt{Ctrl}}$ `W`.
- To open a new text file and with it a new buffer use $\boxed{\texttt{F9}}$. It prompts you for the file name. The buffer name will become equal to the file name.
- To switch the current buffer to another buffer, use $\boxed{\texttt{F10}}$. The buffer name will be prompted.
- To show all available buffers use $\boxed{\texttt{GOLD}}$ $\boxed{\texttt{Select}}$. Select one of them by moving the cursor to the buffer line and use $\boxed{\texttt{Select}}$.
- To move and/or delete blocks, use the following sequence:

  $\boxed{\texttt{Select}}$ or $\boxed{\texttt{.}}$ to enter the select mode,
  move cursor, the selected range is displayed reverse,
  $\boxed{\texttt{Remove}}$ or $\boxed{\texttt{6}}$ the selected range is deleted (moved to paste buffer),
  use $\boxed{\texttt{Insert Here}}$ or $\boxed{\texttt{PF1}}$ $\boxed{\texttt{6}}$ to restore the deleted range if wanted,
  move cursor to the new position,
  $\boxed{\texttt{Insert Here}}$ or $\boxed{\texttt{PF1}}$ $\boxed{\texttt{6}}$ the previous deleted range is inserted.

  The contents of the paste buffer is kept, if you switch between windows or edit buffers. It will only be overwritten by $\boxed{\texttt{6}}$.
- A repetition factor for any key may be entered by $\boxed{\texttt{PF1}}$ and the number (on main keypad), e.g.:

  $\boxed{\texttt{PF1}}$ `10` $\boxed{\texttt{0}}$ to move the cursor 10 lines.

  $\boxed{\texttt{PF1}}$ `10` $\boxed{\texttt{6}}$ to insert the the contents of the paste buffer 10 times.

  $\boxed{\texttt{PF1}}$ `1000` $\boxed{\texttt{PF1}}$ $\boxed{\texttt{Enter}}$ to replace 1000 times the searched strings by the contents of the paste buffer.
- Enter command level by $\boxed{\texttt{Do}}$. The following commands may be useful:

```
SPELL           ! invoke English spell checker for the current buffer

READ file       ! inserts file at cursor position
WRITE file /SEL ! writes selected range to file
WRITE file      ! writes current buffer to file

EXIT            ! exit, save all files
QUIT            ! exit, do not change any file

SHOW BUFFER     ! list of buffer names
```

```
    HELP            ! to get help
```

A shorthand for READ is $\boxed{\texttt{PF1}}$ $\boxed{\texttt{F9}}$ (the filename is prompted).
A shorthand for WRITE/SELECT is $\boxed{\texttt{PF1}}$ $\boxed{\texttt{F10}}$ (the filename is prompted).

- To get a keypad layout, enter $\boxed{\texttt{PF2}}$. To get all keypad definitions, enter $\boxed{\texttt{PF1}}$ $\boxed{\texttt{F7}}$.
- Enter the OpenVMS help by $\boxed{\texttt{F7}}$.
- Recovery: If the Alpha AXP or VAX crashes during an `LSEDIT` session or if you exit by mistake with $\boxed{\texttt{Ctrl}}$ `Y` (**never do that!**), you recover the last session by

  ```
  $ LSEDIT filename /RECOVER
  ```

  NOTE: After test compile (see below), you must recover the file version you started with which is NOT the last one because test compile writes the buffer to the file creating a new version.
- LSEDIT macros can be written in TPU syntax. Examples are in GSI$MANAGER:LSEINIT.TPU.

Besides the screen editing facilities there are some more useful features:

- Language sensitive commands:
  `LSEDIT` recognizes from the file type the kind of file. E.g. if type is `.PPL` or `.PLI`, it assumes a PL/I program, if type is `.FOR` a FORTRAN program and if type is `.COM` a DCL procedure. The language constructs of the language are implemented as **placeholders**, **tokens**, and **procedure calls**. These are inserted in the text enclosed in [ ] or { }. To replace a placeholder just type ahead. Other controlling sequences are:

  $\boxed{\texttt{Ctrl}}$ `E` to expand a placeholder, token, or procedure call
  $\boxed{\texttt{Ctrl}}$ `N` to go to the next placeholder, token, or procedure call
  $\boxed{\texttt{Ctrl}}$ `K` to delete a placeholder, token, or procedure call

  If { } appears, you must insert something. If you type $\boxed{\texttt{Ctrl}}$ `E`, you get a description of what is required here. You can type a placeholder name, press $\boxed{\texttt{Ctrl}}$ `E` and the placeholder will be expanded either to a set of other placeholders or tokens. You get a list of present tokens and placeholders by $\boxed{\texttt{PF19}}$. The best way is to play with this! We implemented some structures for DCL procedures which are not very comfortable in DCL, i.e. IF_ELSE, DO loops and temporary file names: `LOOP` $\boxed{\texttt{Ctrl}}$ `E` expands a DCL DO loop, `IF_ELSE` $\boxed{\texttt{Ctrl}}$ `E` expands a DCL IF-THEN-ELSE construct

- Test compile:
  This is a very useful feature of the `LSEDIT`. Press $\boxed{\texttt{F17}}$ and the current file will be compiled. The error messages are displayed in the top window. You may now correct your source following the messages. **NOTE** that the current buffer is written to its file! $\boxed{\texttt{Ctrl}}$ `N` will skip to the next error, $\boxed{\texttt{Ctrl}}$ `P` will skip to the previous error.

- Include PL/I calling sequence for a module:
  With the the $\boxed{\texttt{F8}}$ key you can include a calling statement for a module. The module name is prompted. This supports all runtime library modules, all system modules, and the GOOSY modules.

- Include text modules from libraries:
  With $\boxed{\texttt{PF1}}$ $\boxed{\texttt{F8}}$ you can include a module from a text library. The library and the module are prompted.

- Execute one DCL line from your text:
  If you have a DCL line in the text your are editing place the cursor in front of this line and press $\boxed{\texttt{F19}}$. The command will be executed as a DCL command and the output will be placed into your text just after the command line.

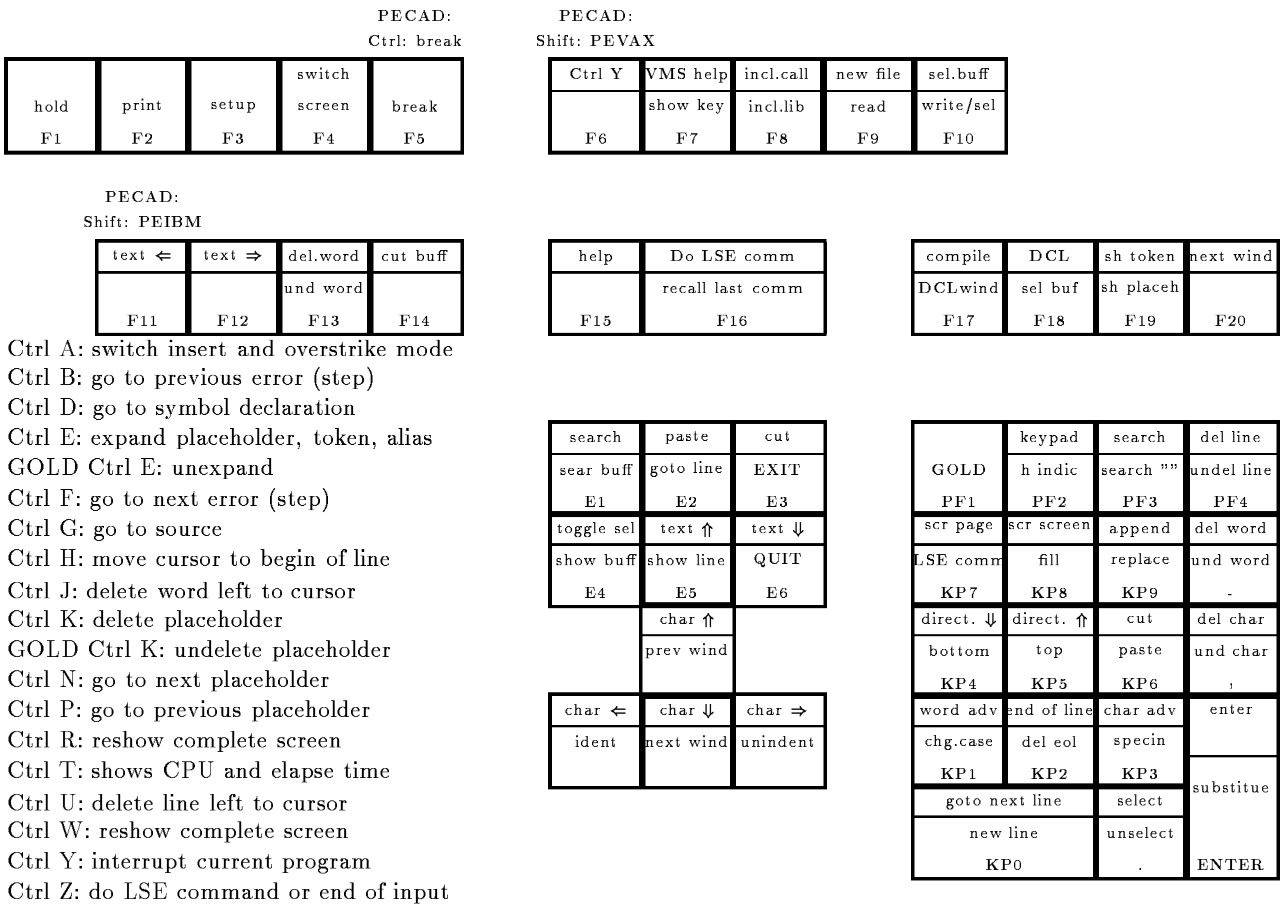
- Execute several DCL lines:
  Press $\boxed{\texttt{GOLD}}$ $\boxed{\texttt{F17}}$ to split your screen. Enter DCL commands to the prompt in the lowest line of the screen. The output will be placed in the upper new window. Leave this mode by just typing $\boxed{\texttt{Return}}$.

- Create another process (SPAWN):
  With the $\boxed{\texttt{F18}}$ key you can execute a DCL line (which is prompted) in a subprocess. After the execution of the DCL command, `LSEDIT` returns immediately to the edit session. If you press $\boxed{\texttt{F18}}$ $\boxed{\texttt{Return}}$ you are in a new, spawned DCL process and you may enter all DCL commands or run any program. To leave this mode and return to your current `LSEDIT` session, type `LOGOUT` on DCL level.

To leave the editor, use $\boxed{\texttt{GOLD}}$ $\boxed{\texttt{Remove}}$ or press DO-key and type `EXIT` $\boxed{\texttt{Return}}$ to exit with writing changed text into a new version of the file or use $\boxed{\texttt{GOLD}}$ $\boxed{\texttt{Next Screen}}$ or press DO-key and type `QUIT` $\boxed{\texttt{Return}}$ to quit (no modifications are saved, no editing in the file is saved, but note that sometimes a buffer is written into its file during a session, e.g. with the compile command!).

In the figure 4.1 on page 38 you see the LSEDIT keypad layout.

The upper key values are the simple key hits, the lower are entered with a preceding GOLD(=PF1)-key hit.



Ctrl A: switch insert and overstrike mode
Ctrl B: go to previous error (step)
Ctrl D: go to symbol declaration
Ctrl E: expand placeholder, token, alias
GOLD Ctrl E: unexpand
Ctrl F: go to next error (step)
Ctrl G: go to source
Ctrl H: move cursor to begin of line
Ctrl J: delete word left to cursor
Ctrl K: delete placeholder
GOLD Ctrl K: undelete placeholder
Ctrl N: go to next placeholder
Ctrl P: go to previous placeholder
Ctrl R: reshow complete screen
Ctrl T: shows CPU and elapse time
Ctrl U: delete line left to cursor
Ctrl W: reshow complete screen
Ctrl Y: interrupt current program
Ctrl Z: do LSE command or end of input
GOLD Ctrl Z: do LSE command or end of input
GOLD =: split/unsplit screen

Figure 4.1: The Special Keypad Layout for the LSEDIT.

4.2 Compiling

You may call the following compilers directly as DCL commands:

```
$ BASIC
$ CC                        ! for the C compiler
$ CXX                       ! for the C++ compiler on AXP only
$ FORTRAN
$ MACRO                     ! for the assembler
$ MODULA                    ! on the VAXes only
$ OPS5
$ PASCAL
$ PLI
```

But the recommended command to call a compiler is

```
$ COMPILE filename.type
```

The compilers are invoked by `$ COMPILE filename.type`, depending on the file type. Standard file types are:

.C = C sources
.FOR = Fortran sources
.MAR = VAX Assembler sources
.MOD = Modula 2 sources
.PAS = Pascal sources
.PLI = PLI sources
.PLITEMP = temporary PLI-source generated by GOOSY preprocessor.
.PPL = GOOSY PLI preprocessor sources

The default type for COMPILE is .PPL, but it can be changed in the login procedure by DEFCOMPI :== <type>. The COMPILE command is also available on IBM for GOOSY preprocessor code. If a file has been compiled already and the source has not changed, it is NOT compiled until the /COM qualifier is specified. Thus one can compile a set of files in a command procedure. Only modified files are compiled saving time. The output of a compilation is (are) object file(s) named like the source file, but with type .OBJ. These files are input for the linker. Some examples:

```
$ COMPILE X             ! compile X.PPL
$ COMPILE X.FOR         ! compile X.FOR
$ COMPILE X,Y           ! compile X.PPL and Y.PPL
$ COMPILE X/DEB         ! compile X.PPL with DEBUG (see below)
$ COMPILE X/COM         ! compile X.PPL again
$ COMPILE               ! compile last set
$ COMPILE X /KEEP       ! do not delete X.PLITEMP (PL/I code)
$ COMPILE X*            ! compile all .PPL files X*
$ COMPILE X/OLB=OPRIV  ! insert object of X in library OPRIV
$ COMPILE X/LIB=TPRIV  ! Search TPRIV for includes
$ COMPILE X /QUALIFIER=(LIST,SHOW=ALL)
                        ! write a file X.LIS containing line numbers
                        ! PL/I switches as shown by HELP PLI must
                        ! be passed to COMPILE by this way
```

You may also compile programs during a LSEDIT session pressing the `F17` key. In case of compilation errors the screen will split showing the error conditions in the upper screen part and the source in the lower part. You may step forwards through the errors pressing `Ctrl` F and `Ctrl` B for stepping backwards. With `Ctrl` G the cursor will move to the faulty source line which can be edited immediately. If a pre-compiler was used, the source displayed is the source after the pre-compilation. In such a case you must edit the original pre-compiler source to eliminate errors. Select the buffer with the original source using `GOLD` `Select`. Unsplit the screen with `GOLD` =.

4.3 Linking

A program must be linked with all called modules to be executed. This is done by the `LINK` command. The modules, except the main program, may be in libraries. Libraries may be specified with the `LINK` command. These are scanned first. Then a list of default libraries is scanned. This list is displayed by

```
$ SLOG LNK$*
```

The output of the link step is a so called executable image, the program in a file which can be executed by the `RUN` command. The default output of the linker is the image file of type `.EXE`. The name is the filename of the main program. Examples for the LINK command:

```
$ LINK X,Y,Z            ! link object files X.OBJ, Y.OBJ, Z.OBJ to X.EXE
$ LINK X,OPRIV/LIBRARY  ! Link object file X.OBJ with modules from OPRIV to X.EXE
$ LINK X,Y/DEBUG        ! Link X.OBJ and Y.OBJ with debugger to X.EXE
```

4.4 Executing

A linked program is executed by

```
$ RUN program
```

Another, more elegant way is to create a symbol 'command' to execute the program:

```
$ command == "$device:[directory]program.EXE"
e.g.
$ BETA == "$KP1$ROOT:[ADAM]BETA.EXE"  ! BETA can be used now as a command
```

This should again be done in the LOGIN.COM file. Then the program is executed by `command`. The LIB$GET_FOREIGN routine called in the program returns any characters typed behind `command`. This is the method to pass parameters to a program together with the execution.

A program can be canceled by `Ctrl` Y. **All utility programs should be rather terminated on terminal input request by** `Ctrl` Z.

4.5 Debugging

The OpenVMS debugger is a very powerful tool to find errors. It allows to set break points on source lines, step line by line, inspect and set variables etc. Modules to be debugged must be compiled with the /DEB option. The main program must be linked with the /DEB option,

```
$ COMPILE program/DEB
$ LINK program/DEB
```

If you run your program with

```
$ RUN program
```

the DEBUG mode is entered by default. If you want to run the same program without DEBUG start it with

```
$ RUN/NODEBUG program
```

If you are under DECwindows/Motif new Debugger windows are popped up. The handling of the DECwindows/Motif Debugger is somewhat unhandy. If you want to suppress the window mode, i.e. you want to run in the text terminal command mode, you must define the following logical

```
$ DEFINE /JOB DBG$DECW$DISPLAY ""
```

If you want to separate the Debugger terminal input and output to another terminal use the DCL command

```
$ DEBWIN
```

It will define the logicals

```
$ DEFINE /JOB DBG$INPUT term:
$ DEFINE /JOB DBG$OUTPUT term:
```

where term: is a valid terminal line, e.g. LTA123:. If you want to have this terminal to be a DECwindows/Motif DECterm window executed the following

```
$ CREATE/TERMINAL/NOPROCESS -
    /WINDOW_ATTR=(TITLE="Debugger",ICON_NAME="Debugger",ROWS=40) -
    /DEFINE_LOGICAL=(TABLE=LNM$JOB,DBG$INPUT,DBG$OUTPUT)
$ ALLOCATE DBG$OUTPUT
```

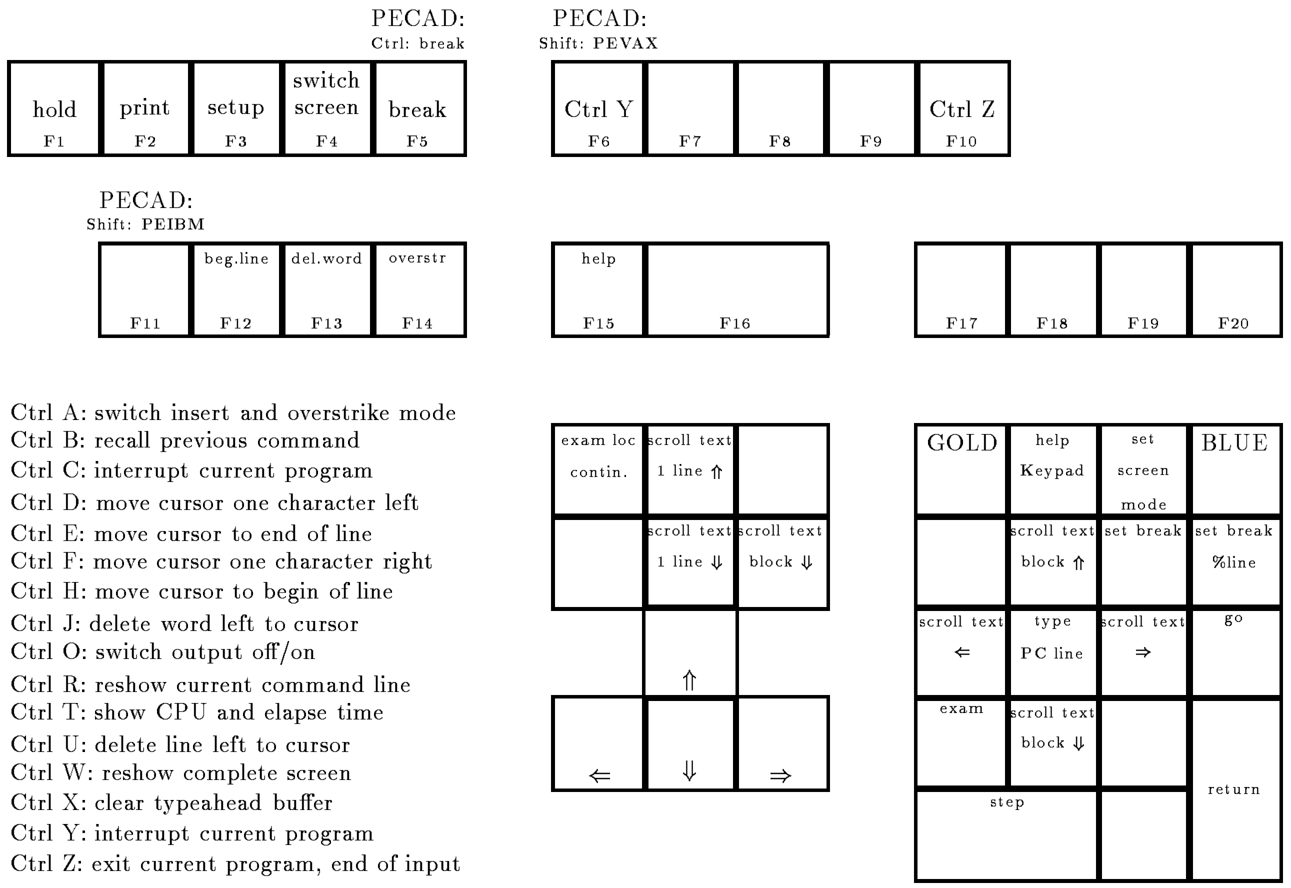
In the following only the text terminal command mode will be described. For the DECwindows/Motif mode refer to on-line Help or the Debugger manual. When you enter the DEBUG menu you may get detailed information with the DEBUG command HELP.

Some most often used commands are shown in the following:

```
DBG> SET BREAK module\%LINE # ! set a break point in line #
DBG> GO                       ! Execute to next break point
DBG> STEP or <KP_0>           ! Execute one line
DBG> EXAMINE variable         ! Examine a variable in decimal
DBG> EXAMINE                  ! Examine the following variable in decimal
DBG> EXAMINE .                ! Examine the last variable again
DBG> EXAMINE @address         ! Examine a location whose address is in 'address'
DBG> EXAMINE /HEX variable    ! Examine a variable in hexadecimal
DBG> EXAMINE /OCT variable    ! Examine a variable in octal
DBG> EXAMINE /ASCII variable  ! Examine a variable as an ASCII character
```

You may examine any single variable, an array, a structure, or a member of a structure. If you examine an array without defining the index limits, e.g. EXAM BETA instead of EXAM BETA(10:20), you will get the contents of all array members. This might be a long procedure if your array is large. You can interrupt the output without leaving the whole debug session by typing $\boxed{\texttt{Ctrl}}$ C). You may although suppress the output by typing $\boxed{\texttt{Ctrl}}$ O but since the debugger continues to examine your array it will still will take some time. **So be careful with the examination of arrays**.

```
DBG> DEPOSIT variable=value   ! Set a variable in decimal
DBG> DEPOSIT variable=%HEX value   ! Set a variable in hexadecimal
DBG> DEPOSIT variable=%OCT value   ! Set a variable in octal
DBG> DEPOSIT variable='string'! Set a variable with a string
DBG> <INSERT HERE>            ! Scroll source up 1 line
DBG> <PREV>                   ! Scroll source down 1 line
DBG> <KP_8>                   ! Scroll source up
DBG> <KP_2>                   ! Scroll source down
DBG> <KP_5>                   ! Return to cursor position
DBG> TYPE #                   ! Type line # of the current module
DBG> TYPE module \#           ! Type line # of module
DBG> SEARCH module string     ! Search string in module text
DBG> SEARCH                   ! Proceed searching the string in module text
```



Ctrl A: switch insert and overstrike mode
Ctrl B: recall previous command
Ctrl C: interrupt current program
Ctrl D: move cursor one character left
Ctrl E: move cursor to end of line
Ctrl F: move cursor one character right
Ctrl H: move cursor to begin of line
Ctrl J: delete word left to cursor
Ctrl O: switch output off/on
Ctrl R: reshow current command line
Ctrl T: show CPU and elapse time
Ctrl U: delete line left to cursor
Ctrl W: reshow complete screen
Ctrl X: clear typeahead buffer
Ctrl Y: interrupt current program
Ctrl Z: exit current program, end of input

More commands are available after pressing the GOLD or BLUE key. Use GOLD Help or BLUE Help to get these keys.

Figure 4.2: The Special Keypad Layout for the OpenVMS DEBUG.

```
DBG> SET LANGUAGE             ! If mixed code of FORTRAN and C set language
DBG> SET SYMBOL               ! If mixed code of FORTRAN and C set symbols
DBG> EDIT                     ! Call the LSEDIT for the current module source
DBG> SPAWN                    ! Spawn a new DCL process, exit it with $ LOGOUT
DBG> <Ctrl>W                  ! Rewrite (refresh) the whole screen
DBG> EXIT or <Ctrl>Z          ! Exit debugger
```

In the figure 4.2 on page 43 you see the DEBUG keypad layout.

4.6 OpenVMS Libraries

The OpenVMS libraries are similar to IBM PDS's. But IBM **members** are called here **modules** (that's live). OpenVMS libraries are supported only by specific commands and utilities. Only libraries of specific types are supported: TEXT, HELP, OBJECT and MACRO libraries. Libraries are created by

```
$ LIBRARY/CREATE/TEXT   filename.tlb
$ LIBRARY/CREATE/HELP   filename.hlb
$ LIBRARY/CREATE/OBJECT filename.olb
$ LIBRARY/CREATE/MACRO  filename.mlb
```

Generally one should define a **logical name** for **any library** file including the device and directory where it resides. Again, these definitions must be done in the LOGIN.COM procedure. The `LIBRARY` command handles libraries. In the following we give most used commands and applications:

HELP Libraries

Used by HELP command.

```
$ LIB/INSERT  library file.HLP     ! insert new help text into library from file
$ LIB/REPLACE library file.HLP     ! replace help text in library from file
$ LIB/EXTRACT=module/OUT=file.HLP library ! extract help text from library
```

The format of the help files is described in Appendix **??**. File type `.HLP` is defaulted.

OBJECT Libraries

Used by LINK command.

```
$ LIB/REPLACE library file.OBJ            ! replace objects from file
$ LIB/EXTRACT=module/OUT=file.OBJ library  ! extract objects from library
```

The module names are determined by the objects in the file. Several objects may be in one file! File type `.OBJ` is defaulted.
To delete modules use for all library types:

```
$ LIBRARY/DELETE=module library
$ LIBRARY/DELETE=(module,module,...) library
```

where module may contain wildcards.

TEXT Libraries

Used for PL/I includes and GOOSY data type declarations compilers and GOOSY.

```
$ LIB/REPLACE library file.TXT            ! replace file as module 'file'
$ LIB/REPLACE/MODULE=name library file.TXT ! replace file as module 'name'
$ LIB/EXTRACT=module/OUT=file.TXT library  ! copy module to file
```

The text in the file / module may have any format. File type `.TEX` is defaulted.

Usage of libraries

Create the following libraries on your master directory (= current after login):

```
$ LIBRARY/CREATE/TEXT PRIV.TLB
$ LIBRARY/CREATE/HELP PRIV.HLB
$ LIBRARY/CREATE/OBJ  PRIV.OLB
```

Add following logical name definitions in your LOGIN.COM:

```
$ DEFINE/JOB TPRIV SYS$LOGIN:PRIV.TLB
$ DEFINE/JOB PLI$LIBRARY TPRIV
$ DEFINE/JOB FOR$LIBRARY TPRIV$ ! optional for FORTRAN
$ DEFINE/JOB HPRIV SYS$LOGIN:PRIV.HLB
$ DEFINE/JOB HLP$LIBRARY HPRIV
$ DEFINE/JOB OPRIV SYS$LOGIN:PRIV.OLB
$ DEFINE/JOB LNK$LIBRARY OPRIV
```

With these definitions, the `HELP` command searches first through your private help library HPRIV, the `COMPILE` command searches include modules first in your TPRIV, and the `LINK` command searches modules first in your OPRIV.

Related Commands

LIBLIS library module
  outputs a list of the modules in the library. The module specification may contain wildcards.

LIBTYPE library module
  outputs the content of the specified modules to the terminal.

LIBSEARCH library module list="search_parameter"
  searches through the specified modules. Search parameters are internally passed to SEARCH command.

LIBCOPY sourcelib module destinationlib module
  copies modules from one library to another.

LIBEXTR sourcelib module
extracts modules from library. Wildcards for module are supported. Module names may be in a text file which must be specified as @file.

LIBDEL sourcelib module
deletes a module from library. Wildcards for module are supported. Module names may be in a text file which must be specified as @file.

4.7 Source Code Analysis SCA

Together with the Language Sensible Editor LSEDIT a Source Code Analyzer SCA is available. SCA is an interactive, multilanguage, source code cross-reference and source code analysis tool that aids developers in understanding large-scale software systems. Because SCA deals with an entire software system, instead of individual modules, it is an effective tool during implementation and maintenance phases of a project. SCA stores compiler-generated information about the set of build sources for querying in one unique location, an SCA library. Thus, SCA is a query tool that allows you to reference and query time-stamped source information that directly corresponds to source modules in your system. When these sources are no longer of value, you can modify or delete the SCA library. The library data generated by supporting OpenVMS compilers consists of names of all of the symbols, modules, and files contained in a specific snapshot of the source. Once SCA libraries are created, you can select a library and query its contents from within LSEDIT, at the DCL level, or via the SCA callable interface. You may do cross-referencing and analysis (locate symbols and their occurrences) and consistency checking (of symbols).

You create a SCA library in the specified user directory by the SCA command

```
SCA> CREATE LIBRARY directory
```

You produce analysis data by using the compile DCL command line of the form

```
$ compiler /ANALYSIS_DATA[=file]  source-file[,...]
e.g.
$ FORTRAN /ANALYSIS_DATA TEST     ! from TEST.FOR produce TEST.OBJ and TEST.ANA
```

For details see DECset or LSEDIT/SCA manuals and Help or the Bookreader or contact the OpenVMS Advisory Service (see names on page **??**).

4.8 DEC Performance and Coverage Analyzer PCA

PCA helps you to produce efficient and reliable applications by analyzing your program's dynamic behavior. PCA also measures codepath coverage within your program so that you can devise tests that exercise all parts of your application.

PCA has two operational components:

1. the collector:
   It gathers performance or test coverage data on a running program and writes that data to a performance data file. You may select either the main image or one of the shareable images in the program's address space. It measures the dynamic behavior of the image you have selected. You may select one or more of the following data:
   - program counter (PC) sampling at fixed sample time
   - CPU sampling at virtual-process sample time
   - counters of the exact number of times that specified program locations are executed
   - coverage data indicating which portions of your program are, or are not, executed during each test run
   - page default data
   - system service data
   - input/output data
2. the analyzer:
   It reads the performance data file produced by the collector and processes the data to produce performance and coverage histograms and tables.

You can run the collector and the analyzer in batch as well as interactively.

To invoke the collector compile all source files you want to analyze with the `/DEBUG` qualifier, e.g.

```
$ FORTRAN /DEBUG TEST
```

Then link the whole program with the `/DEBUG=SYS$LIBRARY:PCA$OBJ` library, e.g.

```
$ LINK /DEBUG=SYS$LIBRARY:PCA$OBJ   TEST
```

Finally run the program, e.g.

```
$ RUN TEST
```

The collector will come up with the prompt

```
PCAC>
```

Now enter PCA collector commands like

```
PCAC> SET DATAFILE TEST
PCAC> SET PC_SAMPLING
PCAC> GO
```

These commands will write the collected data to the file TEST.PCA, will set PC sampling mode and starts the program.

To invoke the analyzer use the DCL command `PCA`

```
$ PCA /COMMAND="command1; command2; ..." data-file
e.g.
$ PCA /COMMAND="SHOW DATAFILE; SHOW LANGUAGE" TEST
```

The analyzer will come up with the prompt

```
PCAA>
```

Now enter PCA analyzer commands like

```
PCAA> NEXT      ! produces a source plot with PC sampling data
```

For details see DEC Performance and Coverage Analyzer PCA manuals and Help or the Bookreader or contact the OpenVMS Advisory Service (see names on page **??**).

4.9 Module and Code Management

A software system can have many program files, object libraries, include files, compilers, and compilation and linking options. The more complex the system, the more difficult it is to reproduce the same program image for each build.

The Module Management System MMS automates and simplifies the building of software systems. It can build simple programs consisting of one or more source files, or complex programs consisting of many source files, message files, and documentation files. It is similar to the `make` command of UNIX.

With MMS, you can specify exactly how a software system is to be built and rebuilt. You do this by using a description file in which you describe the components of the system and the file dependencies used to build and rebuild the system.

Each time you run MMS, it follows the description file you have created, reads the components and dependencies, and builds the same system.

During software development, programmers continually make changes to project files. The Code Management System CMS stores and monitors these files.

CMS allows you to store project files in a central library where they are available to all project members. Some of the tasks you can perform on these files are:

- store files (called elements) in a library
- fetch elements, modify them, and test them in your own directory
- control concurrent modifications to the same element
- Merge concurrent modifications to an element
- create successive versions (called generations) of elements
- compare two generations of an element within a library
- organize related library elements into groups
- define a set of generations of elements as a class to make up a base level or release version of a project
- track which users are working on which elements from the library
- maintain a historical account of element and library transactions

For details see DEC Module Management System manuals and DEC Code Management System manuals or the Bookreader or contact the OpenVMS Advisory Service (see names on page **??**).

4.10 OpenVMS Routines

4.10.1 System Services

OpenVMS provides a large number of system routines for

- Event flag handling.
- Asynchronous system trap handling (AST).
- Logical name handling.
- Input/output
- Process control.
- Time handling.
- Condition (error) handling.
- Memory management.
- Lock manager.

You get a full list and short description by `HELP SYSTEM`. Note, that all routines begin with SYS$... In the HELP, however, they are listed beginning with $... Again, the call of these routines may be inserted by the `LSEDIT` $\boxed{\texttt{F8}}$ key.

4.10.2 Run Time Library Routines

OpenVMS provides a large number of runtime library (RTL) routines for various purposes.

- General library routines (LIB$...).
- Screen handling routines (SMG$...).
- Mathematical routines (MTH$...).
- String handling routines (STR$...).

You get a full list and short description by `HELP RTL`. Again, the call of these routines may be inserted by the `LSEDIT`

4.10.3 Utility Routines

OpenVMS provides program interfaces to several utilities.

- Command language routines (CLI$...).
- File definition routines (FDL$...).
- Library access routines (LBR$...).
- Sort/Merge routines (SOR$...).
- Text processing routines (TPU$...).

For these routines there is still no interactive HELP, but only a manual.

Chapter 5

DCL Procedures

DCL procedures are files containing DCL commands. A DCL procedure can be executed in batch job, or in the current process, or at a remote node.

```
$ @file                       ! Execute interactively.
$ @file /OUTPUT=outfile       ! Execute interactively and write all output
                              !  into the file named 'outfile'
$ SUBMIT/NOPRINT file         ! Submit for batch execution and write all output
                              !  into the log-file named 'file.LOG' under SYS$LOGIN
```

5.1 Command Procedure Format

- Each command line must begin with a dollar sign. Lines without a dollar sign are data lines. To span a command line over several text lines, end with an hyphen and continue next line **without** dollar sign.
- Comments begin with exclamation point, e.g.

  ```
  $! This is a comment
  $ DIRECTORY [USER.TEXT]     ! This is a comment after a DCL command
  ```

- DCL commands should be written in full length to provide clarity.
- Labels are terminated by a colon, e.g..

  ```
  $ GOTO ALPHA
  $ ALPHA:
  ```

- Besides the DCL commands used interactively, there are some used only in procedures:

```
     $ GOTO                 ! Go to label
     $ CALL label           ! Subroutine call
     $ label: SUBROUTINE    ! Subroutine
     $ IF - THEN
       ELSE - ENDIF         ! Conditional execution
     $ INQUIRE              ! Prompt for a string
     $ ON - THEN            ! Control conditions
     $ ON ERROR THEN        ! Control error conditions
     $ ON CONTROL_Y THEN    ! Control Ctrl Y conditions
     $ SET [NO]ON           ! Enable (default) or disable error conditions
     $ SET VERIFY           ! Output commands lines during execution
     $ SET NOVERIFY         ! Do not output commands lines (default)
     $ SYNCHRONIZE          ! Synchronize batch jobs

  ! open an existing file for input and create logical name
     $ OPEN logname file.type

  ! create a new file for output and create logical name
     $ OPEN/WRITE logname file.type

  ! open an existing file for input and output at the start of the file
  ! and create logical name
     $ OPEN/READ/WRITE logname file.type

  ! open an existing file for output at the end of the file
  ! and create logical name
     $ OPEN/APPEND/WRITE logname file.type

  ! close a file, this is NOT done automatically when you leave
  ! the command procedure!!
  ! You cannot open the same file again if you have not closed it before
     $ CLOSE logname

     $ READ logname         ! Read from file
     $ WRITE logname        ! Write to file
     $ EXIT                 ! leave procedure
```

5.2 Command Procedure Variables (Symbols)

You should be aware that DCL is a text interpreter. Therefore DCL does not distinguish between string and numeric variables. Both are represented as symbols. Expressions may be mixed up with symbols having a numeric value and others having a string value, but the result will be unpredictable. String values are enclosed in quotation marks (""). If a symbol is enclosed in apostrophes, it is replaced by its value before the line is processed further. This is important because one can use symbols for very tricky operations. In expressions, or as arguments in lexical functions (see below) the apostrophes are not allowed.

There are **global** and **local** symbols. Local symbols are known only at the procedure level where they are created, whereas global symbols are known to all levels. Global symbols are deleted only by `DELETE/SYMBOL/GLOBAL` or by logoff. You should use only local symbols in DCL procedures.

We try to show with some examples the usage of symbols:

```
$ x1 = "ABC"            ! assign a string to a local symbol
$ x1 = "A""BC"          ! x1 has now the value A"BC
$ x1 = "ABC" + "XYZ"    ! x1 has now the value ABCXYZ
$ x1 = x1 - "A"         ! x1 has now the value BCXYZ
$ x1 = x1 - "X"         ! x1 has now the value BCYZ (one X removed)
$ x1 = 1 + 4            ! x1 has the value 5
$ x1 = "1" + "4"        ! x1 has the value 14
$ y = "ABC" + x1        ! y has the value ABC14
$ 'y' = "X"             ! symbol ABC14 has the value X
$ COPY 'y'.TXT 'y'.TEXT ! copy file ABC14.TXT to ABC14.TEXT
$ x = 5
$ y = 7
$ z = x * y             ! y has the value 35
$ IF x .EQ. y THEN EXIT ! false, do not exit
$ IF x .LT. y           ! nested IF clause
$  THEN
$  ELSE
$ ENDIF                 ! end of IF clause
$ CALL ADAM    ! call the subroutine ADAM
$ ADAM: SUBROUTINE      ! begin of subroutine ADAM
$ ENDSUBROUTINE         ! end of subroutine ADAM
$ r = x .NE. y          ! r has the value 1 (true)
$ IF r THEN GOTO hell   ! true, go to hell
$ IF "A" .LTS. "B" THEN ! true, because A is lower than B
$ x = "r = ''r'"        ! x has the value r = 1 (Note the two '')
$ x = "r = """''r'"""   ! x has the value r = "1"
$ y[0,8] = 27       ! the first 8 bits are set to 27, the escape character
```

5.3 Control Statements

Besides the error control statements (see below) there are only four other control statements:

```
$ IF expression THEN command ! execute command only if expression true

$ IF                         ! Conditional execution
$ THEN
$ ELSE
$ ENDIF

$ GOTO label                 ! Proceed at label
$ label:                     ! label

$ GOSUB ROUT                 ! Proceed at label ROUT, but return
$ ROUT:                      ! label
$ RETURN                     ! return to statement behind GOSUB

$ CALL label                 ! Subroutine call
$ label: SUBROUTINE          ! Subroutine
```

5.4 Terminal I/O

After login, there are the logical names defined already

```
SYS$INPUT          ! data input device (terminal)
                   ! In batch or procedures the file
SYS$OUTPUT         ! output device (terminal)
SYS$ERROR          ! output device (terminal)
                   ! both are in batch jobs the log file
SYS$COMMAND        ! command input device (terminal)
                   ! in batch defined as disk
```

To write a line on the terminal use

```
$ WRITE SYS$OUTPUT "any text"
$ WRITE SYS$OUTPUT symbol
$ WRITE SYS$OUTPUT "any text and ''symbol'"
```

To read a line from the terminal into any symbol use

```
$ INQUIRE symbol "any prompt text"
```

The string defined with the INQUIRE statement will be typed on the terminal first (prompt string) followed by a colon ":". If you want to suppress the colon use INQUIRE/NOPUNCTATION.

DCL reads commands from SYS$COMMAND and writes output to SYS$OUTPUT or SYS$ERROR. Any prompts from programs are read from SYS$INPUT which is the DCL procedure file. You must provide the input as data lines behind the line calling the program. If you want the program to get the input from your terminal, add the following line **before** the line calling the program:

```
$ RUN program       ! program reads input from following lines
1234567
any text or numbers
$ DEFINE/USER SYS$INPUT SYS$COMMAND
$ RUN program       ! program reads input from terminal
$ EXIT
```

Of cause, it cannot make sense to execute such procedures in batch jobs, because the program cannot get an input.

5.5 Command Procedure Parameters

Calling a DCL procedure one may specify parameters. These are separated by spaces or (Tab characters. They are converted to uppercase, unless they are enclosed in quotation marks. Inside the procedure these parameters are assigned to local symbols P1 through P8. They may be used as other symbols.

Because this is not a very comfortable parameter passing mechanism, we provide an interface which allows similar parameter lists as for DCL commands. In addition it generates a menu on request. The example shows how to use this interface. A more detailed description is found by HELP MDCLLIST.

```
$ descriptor = "PREF A=? B=DEV C "
$ qualifier  = "/SWI*TCH/VAL*UE=/DEF*AULT=XXX"
$ MDCLLIST_help = "Short description of the procedure."
$ MDCLLIST "''descriptor'" + "''qualifier'"
$ IF .NOT. $STATUS THEN EXIT
$! Now the following symbols are created
$! PREF_A has value of first positional parameter or is prompted
$! PREF_B has value of second positional parameter or DEV
$! PREF_C has value of third positional parameter or null string
$! PREF_SWITCH has value "/SWITCH" or null string
$! PREF_VALUE has value specified in argument list or null string
$! PREF_DEFAULT has value specified in argument list or XXX
$!
```

```
$! This routine may be called as
$! @file ?                           ! Enter menu
$! @file X                           ! 1 positional parameter
$! @file X Y/SWI                     ! 2 pos. par. and a qualifier
$! @file X /VAL=1
$! @file X "" Z /VAL=1/SWI /DEF="A B"! second pos.par is null string
```

It is strongly recommended to create a symbol for the DCL procedure call:

```
symbol == "@device:[directory]procedure"
```

The device and directory specification is required, because the procedure might be called from a different device/directory than the current. Called by a symbol the above procedure calls would look like DCL commands! **Before you create global symbols, check, if they are already in use** (command `SSYM symbol`).

5.6 Lexical Functions

The real strength of DCL are the **lexical functions**. These are functions beginning with F$. They are treated like symbols (but you cannot assign a value to them). We give a short overview and some examples here. You get information by `HELP LEXICAL`.

- **The following functions return information:**

```
F$CONTEXT         ! specifies selection criteria for F$PID use
F$CSID            ! returns cluster identification number
F$DEVICE          ! all specific devices
F$DIRECTORY()     ! current default directory
F$ENVIRONMENT     ! information about current environment
F$FILE_ATTRIBUTES ! file attributes
F$GETDVI          ! device information
F$GETJPI          ! process information
F$GETQUI          ! queue information
F$GETSYI          ! system parameters
F$MESSAGE         ! Text associated with message code
F$MODE()          ! execution mode (batch, interactive etc.)
F$PID             ! process ID's
F$PRIVILEGE       ! privileges of current process
F$PROCESS()       ! name of current process
F$SETPRV          ! returns state of queried privilege
F$TIME()          ! time as dd-mmm-yyyy hh:mm:ss.cc
F$TRNLNM          ! translate logical name equivalent
F$TYPE            ! type of a symbol, INTEGER or STRING
```

```
F$USER()          ! user identification code (UIC)
F$VERIFY          ! verification status
```

Examples:

```
$ old_default = F$DIRECTORY()  ! save directory
$ SET DEFAULT [X.Y]            ! set directory
$ SET DEFAULT 'old_default'    ! restore directory

$ IF F$MODE() .EQS. "BATCH" THEN SET VERIFY
                               ! set verification ON in batch only
$ IF F$MODE() .EQS. "BATCH" THEN -
  WRITE SYS$OUTPUT "Job executed at ''F$TIME()'"
                               ! write message to log file
$ WRITE SYS$OUTPUT "You are on terminal ''F$TRNLNM("SYS$COMMAND")'"
                               ! output terminal device
$ WRITE SYS$OUTPUT F$GETSYI("HW_NAME") ! type computer model name, e.g.
Digital 2100 Server Model A500MP
$ WRITE SYS$OUTPUT F$GETSYI("ARCH_NAME") ! type architecture name, e.g.
Alpha
$ IF F$GETSYI("ARCH_TYPE") .EQS. "2" THEN GOTO ALPHA_CODE
```

- **The following functions handle strings:**

```
F$EDIT            ! edit a string
F$ELEMENT         ! get elements out of a string separated by a delimiter
F$EXTRACT         ! extract a substring
F$FAO             ! format a string with arguments
F$LENGTH          ! length of a string
F$LOCATE          ! locate a substring
```

Examples:

```
$ x = "ABCDEFG"            ! assign value
$ i = F$LOCATE("C",x)      ! i is 2 (offset!)
$ l = F$LENGTH(x) - i      ! length minus offset
$ y = F$EXTRACT(i,l,x)     ! y is CDEFG

$ x = "A,B,C"
$ y = F$ELEMENT(0,",")     ! y is A, delimiter is comma
$ y = F$ELEMENT(3,",")     ! y is comma (no element of this number)
$ x = "   a    "
$ y = F$EDIT(x,"TRIM")     ! y is a
$ y = F$EDIT(y,"UPCASE")   ! y is A
$ x = "a   a   a "         ! several spaces
$ y = F$EDIT(x,"COMPRESS") ! y is a a a (one space)

$ x = "This is !AS"        ! The !AS is a dummy for a string
$ y = F$FAO(x,"A")         ! y is This is A
```

- **The following functions convert data types:**

```
F$CVSI              ! extract bit fields and convert to integer
F$CVTIME            ! convert time to yyyy-mmm-dd hh:mm:ss.cc
                    ! this time format can be compared to another
F$CVUI              ! extract bit fields and convert to integer
F$IDENTIFIER        ! converts an UIC identifier
F$INTEGER           ! converts expression to integer
F$STRING            ! converts expression to string
```

- **The following functions handle file names:**

```
F$PARSE             ! parse file names, replace defaults
F$SEARCH            ! search a file, return full name
```

Examples:

```
$ file = "LOGIN"
$ def  = F$PARSE(file)            ! e.g. EE$ROOT:[GOOFY]LOGIN.;
$ def  = F$PARSE(file,".COM")     ! e.g. EE$ROOT:[GOOFY]LOGIN.COM;
$ dir  = F$PARSE(file,,,"DEVICE") ! e.g. EE$ROOT:
$ full = F$SEARCH(file)           ! e.g. null string (LOGIN is not found)
$ full = F$SEARCH("LOGIN.COM")    ! e.g. EE$ROOT:[GOOFY]LOGIN.COM;53
```

As you can see the file type and the version number are not defaulted.

5.7 Command Procedure Debugging

DCL itself allows only to set verification on (`SET VERIFY`) and to place `SHOW SYMBOL` statements in the procedure to find out what happens. At GSI there is a tool to generate a debug version of a procedure without modifying the run version:

```
$ MDCLANAL DEBUG procedure debugfile    ! generates debug version
```

After that, debugfile contains a version of the procedure showing all symbols. You may specify, if verification is switched off during procedure calls or not. This is very convenient, if you call other, already tested procedures.

5.8 Command Procedure Error Handling

You should provide generally one or more labels where errors are handled and one where breaks (Ctrl Y) are handled. You may close files, delete temporary files, give error explanations etc...

```
$ ON ERROR THEN     GOTO error_label
$ ON CONTROL_Y THEN GOTO break_label
$ SET NOON              ! disable error checking
$ SET ON                ! enable error checking
```

After execution of programs you may check the symbol $STATUS for a successful completion ($STATUS = 1), e.g.

```
$ IF $STATUS .NE. 1 THEN GOTO ERRORHANDLING
```

5.9 Read/Write Files in Command Procedures

It is very easy to create, read and write text files. The following examples show that. The parameter 'logname' is an automatically defined logical name used to connect read/write statements to open/close statements.

```
   $ OPEN/WRITE/ERROR=label logname file  ! create new file
   $ OPEN/APPEND/ERROR=label logname file ! open existing file for write
   $ OPEN/READ/ERROR=label logname file   ! open existing file for read
   $ READ/ERROR=label1/END_OF_FILE=label2 logname symbol
                                          ! read record to symbol
   $ WRITE logname expression             ! write expression to record of file

! if file is NOT closed automatically when you leave the command procedure!!
!  you cannot open the same file again if you have not closed it before
   $ CLOSE logname                        ! close that file
```

Please, specify generally labels to handle errors or end of files. Otherwise files would stay OPEN and cannot be deleted!

5.10 LSEDIT Support for Command Procedures

Some unhandy DCL constructs are generated by the LSEDIT editor. These constructs are inserted in the text typing one of the following names terminated by Ctrl E

```
IF        ! IF statement
IF_ELSE   ! IF ELSE construct
```

```
IF_END     ! IF END construct
LOOP       ! Loop setup
TMP        ! unique temporary file name
```

Remember, that you have to replace 'placeholders' and 'tokens' marked by [] or {}, respectively. To do that, just type. Go to the next one by $\boxed{\texttt{Ctrl}}$ N and replace it and so on.

Part II

GOOSY Introduction

Chapter 6

GOOSY Introduction

6.1 GOOSY Summary

6.1.1 Summary

GOOSY is a PL/1 based nuclear data acquisition and analysis system implemented on VAX/AXP OpenVMS computers. It gets input from MBD, CAMAC processors (J11, CVC), or VME processors reading out CAMAC modules and/or Fastbus subsystems. The analysis is done by user written routines and/or command controlled analysis tables. Network capabilities support several analysis programs on different DECnet nodes and several displays. A data management allows for comfortable control of PL/1 structured Data Elements. Among others, the main features of GOOSY are:

1. GOOSY is composed of loosely coupled programs like Transport Manager, Display, Analysis, and Data Manager. This structure provides high security and flexibility because the data acquisition is independent of the analysis.

2. The programs may run on different VAX/AXP computers connected via DECnet. The experiment can be controlled from several terminals.

3. The analysis is independent of input which may come from a Transport Manager (CAMAC) via mailbox or DECnet, another analysis via DECnet, or disk or tape.

4. The analysis can be done by user written routines utilizing macros and/or by analysis tables which can be modified and/or (de)activated by interactive commands. The analysis tables control the following actions:

   - check of conditions (window, multiwindow, pattern, boolean, polygon and user defined function conditions) or condition arrays,
   - accumulation of histograms (1- and 2-dimensional, real, integer, and bit spectra with condition filters) or histogram arrays controlled by multiwindow conditions or condition arrays,
   - user written procedures filtered by conditions,
   - density distributions of pairs of any parameters filtered by conditions.

5. The Transport Manager presently supports the parallel CAMAC branch driver MBD and a J11 based auxiliary crate controller CES-2180 Starburst with an Ethernet/DECnet connection. It supports a VME based multiprocessor frontend system controlling CAMAC, Fastbus or foreign subsystems. A CAMAC computer board (CVC) providing a VSB bus to connect CAMAC and/or Fastbus will also be supported. A standard program interface is provided to support other frontend equipment.

6. The GOOSY display provides various tools to handle complex display pictures composed of histograms and distributions.

7. All data of GOOSY (e.g. spectra, conditions, analysis tables, pictures, event data, user defined Data Elements) are stored in Data Bases preserved independently of programs even in case of crashes. They are handled by a Data Manager Program.

8. The analysis of one data stream can be done by several analysis programs running on different DECnet nodes in parallel.

6.1.2 GOOSY Subsystems

Data Management

In a software system for data acquisition and data analysis a large number of differently structured data objects has to be handled. Those data objects could be simple variables like coordinates or calibration parameters or complex structures like a spectrum or a picture. Therefore a Data Base system has been developed and implemented. Data Bases are organized in Directories, similar to OpenVMS, the VAX operating system. All Data Elements have names which are inserted in Directories. Various commands are provided to handle any kind of Data Elements, as CREATE, DELETE, SHOW, COPY, SET. Data Element specifications have the general form:

```
node::base:[directory]name(index).member(index)
```

The node name will be used in the future for remote Data Base access. All Data Elements are structures like PL/1 variables. In addition - and in contrast to a file system - a Data Element name can be indexed. Each member of such an array has the same data structure. The smallest referable entity of a Data Element is any member of its structure.

The Data Base is split into subdivisions called Pools. A Pool is the smallest part of a Data Base which can be mapped into a program's address space where it is contiguous. The mapping specifies the program's access rights to the Pool, e.g. read only or read/write. A Pool is a chain of Data Areas. Areas are contiguous in the file. If the space in a Pool is exhausted, one more Area is added to the chain thus expanding the Pool. All Data Elements are allocated in Pools. All allocations and releases in a Data Base are logged in bit maps. Thus released space can be allocated again.

Any Data Element in the Data Base is described by a Type descriptor. This is used to handle the Data Element, e.g. for CREATE, COPY and SHOW commands. The Type descriptor is generated from a PL/1 structure declaration which can be included in a procedure to access the Data Element. Thus the consistency of the access is provided. The local pointer to the Data Element is returned by locating procedures or macros. After locating the Data Element is protected against deletion.

Several programs can work independently with the same Data Base thus sharing data. Therefore some actions are locked to ensure the integrity of the Data Base and the local mapping contexts in all attached programs.

Command Interface

The GOOSY command syntax is similar to DCL. The commands are not implemented directly in DCL because of conflicting command keywords. They are rather implemented like commands available in DEC utility programs. They can be used in DCL procedures. Through a prompter program commands can be dispatched to subprocesses where they are executed. The commands are defined by subroutine calls. Each command is executed by a PL/1 or FORTRAN routine. Commands are composed of several keywords, e.g.

```
START INPUT NET
START INPUT MAILBOX
START OUTPUT FILE
```

A convenient command input is provided by a menu option. The menu is generated automatically from the command definition. On each menu level interactive help is available. The command parameters can be input in a positional order, by name reference or as qualifiers:

```
CREATE SPECTRUM/DIGITAL LIMITS=(1,1024) DATATYPE=L NAME=name
CREATE SPECTRUM name L (1,1204) /DIGITAL
```

By a logical name mechanism the user can assign names to parts of a command line thus simplifying input of often used commands. Certain parameter values can be preset in profiles. The parameter defaults are optionally updated. Other options include command files executed by the interface and recalling of command lines.

The interface module may be called several times in user written subsystems.

Menu Driven Guide

For users who want to use GOOSY without using manuals or HELP facilities a menu driven GOOSY guide provides a very easy to use interface to GOOSY. Most DCL procedures also have a menu interface.

Graphic System

The GOOSY graphics system is realized with GKS (Graphical Kernel System) which allowed to design the graphic package independently of the graphical output devices. Supported device types are among others Motif windows, postscript files and LN03-PLUS laser printers. A device independent plotfile can be generated which can be re-displayed on any graphical device. This file could be sent to IBM to be plotted on all plotters available.

To control complex experimental set-ups with many detector sub-systems, as planned for SIS/ESR experiments it is necessary to display the spectra for several of those sub-systems at the same time. Therefore the GOOSY graphic system provides a tool to summarize up to 64 frames for spectra and/or scatterplots in pictures. The size of the spectra and scatterplots in the frames and the organization of the frames on the screen can be defined by the user.

The spectra in the frames can be displayed as histograms, in vector or marker mode (for 1-dimensional spectra), and as contour, isometric, or cluster-plots (for 2-dimensional spectra) in linear, logarithmic or square-root mode on any axis. For the contour and cluster-plots arbitrary numbers of cuts are definable. The isometric plot of a two-dimensional spectrum can be rotated in any direction.

Some other features provided by the GOOSY graphics system are:

1. PLOT commands to send the actual picture on the screen or any created metafile to the plotters at VAX or IBM.
2. Update of each frame on the screen in an arbitrary time interval.
3. Projection of two-dimensional spectra.
4. Fit of background and gaussian peaks.
5. Overlays of spectra or additional scatterplot parameters in the corresponding frames.
6. Zooming of a single frame out of the actual picture on the screen.
7. Calibrations of spectra or axis labelling.

6.1.3 GOOSY Components

GOOSY components are programs executing commands. These programs can be started from the VAX/OpenVMS command language DCL or run in subprocesses. In the first case, they get the commands from the terminal, in the second case from a prompter program. Thus several components can be controlled from one terminal. The set of components controlled from one terminal is called 'environment'. Each environment has a supervisor program to dispatch messages between the prompter program and the components.

GOOSY Prompter

This program is started to control components running in an environment. It executes the following functions:

- Create/Delete environment.
- Create/Delete processes.
- Dispatch Commands.

Supervisor and components are started as subprocesses. GOOSY commands are dispatched via the supervisor to the subprocess where the command is executed. The prompter hibernates until it gets an acknowledge message from the supervisor. Therefore the subprocess executing the command can use the terminal for I/O.

Transport Manager

This is the central data dispatcher. It controls the frontend equipment, gets data buffers and dispatches them to DECnet channels, mailboxes and files (disk or tape). Presently it supports CAMAC by the MBD branch driver or by a single crate controller J11. A new CAMAC board utilizing a fully equipped 68030 processor can control other CAMAC or Fastbus crates through a differential VSB bus. A powerful VME frontend system allows to control CAMAC, Fastbus and other subsystems. If other frontend equipment will be needed in the future, this is the only interface which has to be extended. A standard program interface is provided to support other frontend equipment. Because this program runs independently of other programs, data writing is not interrupted by errors occuring in analysis or display programs.

Data Base Manager

This program executes all commands to maintain Data Bases and Data Elements, like CREATE, DELETE, SHOW, MODIFY, COPY etc.

Display

This program executes all display commands. It allocates the output device. It gets data from analysis programs via DECnet for scatter plots.

Analysis

This user specific program analyzes the data. It may get data buffers from DECnet, Mailbox or files (disk or tape). It may output data buffers to DECnet and files. During an experiment several analysis programs may run on different nodes getting the same data from one Transport Manager. Analysis programs can also be chained. The analysis program need not to be modified for online or offline mode.

Others

More programs may be started at any time. They may execute commands and control Data Elements or hardware components.

6.1.4 GOOSY Data Elements

As mentioned above, all data are located in Data Bases. There are some Data Elements defined by GOOSY which are handled by special commands. These are described in the next sections.

Condition

All kinds of conditions can be executed through analysis tables (see below). They may then be used as filters for spectrum accumulation and/or scatter plots. In an analysis routine they are executed by the macro $COND. Each condition has TRUE and FALSE counters and bits for freeze, execute, result, and preset. The different kinds are:

- Window Condition
- Multiwindow Condition
- Pattern Condition
- Function Condition
- Polygon Condition
- Composed (boolean) Condition

Polygon

Polygons may be created, displayed, modified, copied, and deleted. They can be specified by graphic input or numerically. They are used by one or more conditions.

Spectrum

Spectra may be of the type integer or float. The dimensionality can be one or two. Spectra can be filled through analysis tables or by the macro $ACCU.

Calibration

Calibrations are tables used to calibrate the spectra data when displaying them. Each calibration can be connected to several spectra by the `CALIBRATE SPECTRUM` command (see section 6.12.3 on page 126).

Picture

Pictures keep information to display up to 64 frames containing spectra or scatterplots on one screen.

User Defined Data Elements

Besides the GOOSY Data Elements the user may define and create his own Data Elements. This may be done by GOOSY commands or by subroutine calls in a program. To access the Data Element in a program, include the library module declaring its structure and call $LOC macro to receive the pointer to the Data Element.

6.1.5 Supported Hardware

Presently two CAMAC setups are supported by the Transport Manager:

1. Up to seven CAMAC crates each controlled by J11-based auxiliary crate controllers connected to a VAX through an MBD (Microprogrammed branch driver, BiRa Systems Inc.)

2. Single CAMAC crate system controlled by a J11 (PDP-11/73 in a CAMAC module, CES 2180 Starburst) connected to a VAX/AXP through DECnet.

The two subsystems are replaced by a VME based frontend system and a new CAMAC computer system:

1. VME frontend system. Utilizes 68020 processors, each controlling a VSB branch. The VSB branches connect to CAMAC, Fastbus or other subsystems. One processor collects the data and builds the events, another one does all the communication. The event builder transfers the data to a VAX through a parallel interface or through ethernet.

2. CAMAC computer. The board is fully equipped with a 68030 processor, ethernet, SCSI and VSB. Similar to the VME system, different kinds of subsystems may be connected to the VSB. The data is supposed to be written to a local tape.

With all setups, CAMAC commands can be executed (ESONE calls).

J11

For smaller experiments with lower data rates a CAMAC single crate system is supported by GOOSY. A J11 running RSX-11S reads out the CAMAC modules, fills compressed events into buffers which are sent via DECnet to a Transport Manager on a VAX/AXP. The CAMAC setup is described by a text file provided by the user. No programming work is required.

MBD

Each J11 crate controller reads out the CAMAC modules of one crate. The MBD copies the compressed subevents from the J11's to event buffers which are then sent to the VAX memory. Programs for the MBD are provided for standard setups. The stand alone software for the J11's can be generated from CAMAC description files and/or written and modified by the user. In the analysis program the event unpack routine may be written by the user as well.

This setup provides high flexibility and performance.

VME

The VME based frontend system is a multiprocessor system allowing very complex structures. Various kinds of subsystem can be connected through a VSB interface. The readout and processing is done in parallel. The system is fully controlled by the GOOSY transport manager.

CAMAC CVC

This is the latest development. The CAMAC board runs Lynx on a 68030 processor. Via VSB other subsystems can be connected. This system will be able to run standalone. A tape can be connected to the SCSI bus. For monitoring purposes selected data can be sent via ethernet (TCP) to GOOSY and/or PAW analysis sessions.

6.1.6 GOOSY Analysis

Event Loop

GOOSY processes experimental data structured in events. The routine executing the event loop is provided by GOOSY. It executes the following steps:

- Call unpack routine (Get new buffer if no more event is found).
- Call analysis routine (optional).
- Call analysis table executor (optional).
- Call standard routine to pack event into output buffer (Output buffer if it is filled) (optional).

The user provides the unpack routine and optionally an analysis routine.

User Written Analysis Routine

GOOSY provides PL/1 macros to locate Data Elements, accumulate spectra and check conditions. The user may write his own analysis routine using these macros. Any Data Element in a Data Base can be accessed. They are declared as PL/1 based structures. The pointers to the structures are set by the macros.

Dynamic Analysis

To improve the flexibility of the analysis GOOSY provides an analysis table executor. It executes condition checks, spectrum accumulations and scatter plots controlled by analysis tables. Commands are provided to edit the analysis tables. Each Entry in an analysis table is composed of the name of a subject Data Element and a list of names of object Data Elements, i.e. a spectrum name as subject and a member of a Data Element structure as object (optionally an additional condition name). The Entries can be added, deleted or modified without effect to a running analysis program. The modified table can then be activated in the analysis program by a command. Of course, spectra and conditions used in analysis tables can be created independently of any analysis program. The analysis table executor presently processes the following types of Entries, in the order as listed:

Function A user written function is called.

Function Condition The condition value is evaluated by a user written function.

Window Condition The values of the specified objects are checked versus the limits of the window.

Multiwindow Condition The value of the specified object is checked versus the limits of the window.

Pattern Condition The value of the specified object is checked versus the pattern of the condition.

Composed Condition This is the result of a boolean expression of other conditions.

Accumulation Supported are 1- and 2-dimensional spectra of type INTEGER or FLOAT.

Scatter Plots The Entry for scatter plots is added to the analysis table by the DISPLAY command, if the displayed picture contains any scatter plot frame.

Any of the conditions may be used as filters for either spectrum accumulation or for individual scatter plot frames. A condition can be inserted as master condition. If a master condition is false, the table execution is terminated.

6.2 Prerequisites for GOOSY

If you never have used GOOSY before, some preparations have to be done. You must be familiar with the basic VAX/OpenVMS concepts and commands (see VAX/OpenVMS Introduction manual).

6.2.1 Account

Ask the GOOSY group if your account is sufficiently priviledged to run GOOSY.

6.2.2 Information Mailer and VAX notes

Ask the GOOSY group to be included in the GOOSY information mailer list. You can list all GOOSY mails on the terminal by

```
$ GNEWS
```

All mails also are written to VAX notes. The users also can write comments to VAX notes. Thus, VAX notes should be used for communication amongst the users and the GOOSY group. A simple interface to VAX notes is provided by command

```
$ GNOTES ?    ! Enter menu
$ GNOTES WRITE "Titel" /USER/KEY=ERROR/FILE=report.txt
```

The second line inserts the content of the file into the conference GOOSY-USER. See `HELP GNOTES` for more information.

6.2.3 Creating Libraries

Create the following libraries by the commands (after you logged in):

```
$ LIB/CRE/TEXT PRIVLIB.TLB
$ LIB/CRE/HELP PRIVLIB.HLB
$ LIB/CRE/OBJ  PRIVLIB.OLB
```

6.2.4 LOGIN Procedure

In the LOGIN.COM procedure some setups for GOOSY must be done.
Copy GOO$EXE:USER_LOGIN.COM to LOGIN.COM and insert your specific statements at the marked places. (Execute the procedure after this).

```
$ SET NOVERIFY
$! Define names for libraries:
$ DEFINE/JOB PLI$LIBRARY SYS$LOGIN:privlib.TLB
$ DEFINE/JOB tpriv SYS$LOGIN:privlib.TLB
$ DEFINE/JOB LNK$LIBRARY SYS$LOGIN:privlib.OLB
$ DEFINE/JOB opriv SYS$LOGIN:privlib.OLB
$ DEFINE/JOB HLP$LIBRARY SYS$LOGIN:privlib.HLB
$ DEFINE/JOB hpriv SYS$LOGIN:privlib.HLB
$! Define names for profiles:
$ DEFINE/JOB GOO$PROFILE GOO$EXE:PROFILE.PROF
$ DEFINE/JOB GOO$INI_ALL GOO$EXE:INI_ALL.COM
$ DEFINE/JOB GOO$INI_TPO GOO$EXE:INI_TPO.COM
$! Set DCL function keys (Press PF2 for help):
$ PFKEY
$! Set system prompt to "node:user$ ":
$ @GOO$EXE:SETPROMPT.COM
$! Define all GOOSY stuff:
$ @GOO$EXE:GOOLOG.COM
$! Establish a Data Base for analysis control:
$ GOOCONTROL
$!
$! Add here user specific statements:
$!
$ IF F$MODE() .NES. "INTERACTIVE" THEN GOTO G_BATCH
$!
$! Add here statements to be executed interactively only:
$!
$!
$ G_BATCH:
$ IF F$MODE() .NES. "BATCH" THEN EXIT
$ WRITE SYS$OUTPUT "******** Starting user batch procedure ********"
$!
$! Add here statements to be executed in batch only:
$!
$ SET VERIFY
$ EXIT
```

6.2.5 Profiles

In profiles one may set default values for certain command parameters. We recommend to use the default profiles. Define in your LOGIN procedure (as shown above):

```
$ DEFINE/JOB GOO$PROFILE GOO$EXE:PROFILE.PROF
$ DEFINE/JOB GOO$INI_ALL GOO$EXE:INI_ALL.COM
$ DEFINE/JOB GOO$INI_TPO GOO$EXE:INI_TPO.COM
```

You may, however, copy these files to your Directory, modify them and define the logical names to your private profiles.

6.3 GOOSY Command Interface

6.3.1 GOOSY Guide

The easiest way to start with GOOSY is to use the guide. The guide is invoked by

```
$ GUIDE GOOSY
```

The $\boxed{\texttt{KP\_0}}$ on the keyboard calls the guide, too. The guide displays menus with brief descriptions of tasks. These tasks are executed by entering the task number as displayed. The guide is completely self describing. The actual GOOSY commands executed are displayed. So one can learn from the guide by using it. All steps except the programming of analysis routines and the prerequisites can be done in the guide. One may leave and reenter the guide at any time. The top level menu looks like this:

```
     Guide to GOOSY: path = ,  last topic =

     ---- GOOSY top menu  ----------------------------------------------

        1- Preparations needed before using GOOSY
        2- GOOSY operating. Everything to control a running GOOSY.
        3- Mostly used SHOW commands
        4- Spectrum handling
        5- Picture handling
        6- Condition handling
        7- Scatter plot handling
        8- Mostly used display commands
        9  Overview of GOOSY commands (HELP)
       10  Reset defaults to values from profile
       11- Enter menus of GOOSY components
       12  Give an overview over the current GOOSY
       13  Shutdown GOOSY

 <RET>, <CTRL>Z, B: Previous menu  <CTRL>Y, E: Exit  ?, H: Help  R: Refresh

  Enter number to select topic :
```

The topic numbers with a hyphen (-) indicate other menus, all others execute DCL procedures prompting for all necessary information.

6.3.2 Line Input

A GOOSY command line can be given in the following situations (environment and components are described later in this introduction):

1. Input to a GOOSY component prompt, e.g. `SUC: DBM>`. The component must have been started 'stand alone' on DCL level, e.g. by `$ MDBM`.

   ```
   $ MDBM
   SUC: DBM> command
   ```

2. Input to the GOOSY prompter (`SUC: GOOSY>`). A GOOSY environment and the appropriate component must have been created.

   ```
   $ GOOSY
   SUC: GOOSY> command
   ```

3. Input to DCL prompt. The GOOSY command must be preceded by the component name, e.g. `MDBM` or by `GOOSY`. The first case is equivalent to (1), the second to (2). The difference is that the command interface returns to DCL level. In addition it is possible to use DCL symbols in the command line.

   ```
   $ MDBM command      ! execute one Data Base Manager command
   $ GOOSY command     ! Excute one GOOSY command
   ```

4. In a GOOSY menu after pressing $\boxed{\texttt{KP\_PF3}}$ key.

Parameters and Qualifiers

The GOOSY command syntax is similar to the DCL syntax. A command is composed of several keywords. Positional parameters are delimited by spaces. String parameters containing delimiters must be enclosed in quotes ("string"). Integer parameters can be entered in decimal, hexadecimal (%Xn), octal (%On), or bit (%Bn) format. In addition to DCL, however, the parameters have names and may be specified in any order by `name=value`. The names are found in the help description of the command or in the menu.

Qualifiers are preceded by a slash (`/qualifier`). Differently to DCL they cannot specifiy a value. In most cases qualifiers can be negated by `/NOqualifier`. Qualifier sets are mutually exclusive qualifiers. In the menu the positional parameters are specified on top together with their names, followed by the qualifiers. At last, the qualifier sets follow together with their qualifiers. An example is shown below.

Command Procedures

The GOOSY command interface is able to process command files (default file type is .GCOM). These files contain GOOSY commands lines. The commands in these files are executed by the '@' command:

```
$ GOOSY @file
$ MDBM @file /LOG
```

Comment lines may be written starting the line with an exclamation point. Lines can be continued by ending with a hyphen (-). Optional arguments replace strings &P1...&P8 in the command lines.

Defaults

In some cases default values are provided for command parameters. Some parameter values are stored as new defaults, if specified. Global defaults are valid for several commands, e.g. the Data Base name. Note, however, that the scope of global defaults is limited to a component (see 6.9.3). Only global defaults can be preset in the profiles. Some parameters are required, i.e. they are prompted if not specified. In the command description these attributes are described for each parameter:

```
parameter  default: value
parameter  replace
parameter  global replace
parameter  required
```

Command Example

The following command has two positional parameters, one switch and one qualifier set. Command description:

```
SET ACQUISITION in_buffers out_buffers
                /[NO]SYNC
                /MAILBOX/NET
```

Valid commands are then

```
SET ACQ 4 2 /SYNC/NET  ! in_buffers=4, out_buffers=2
SET ACQ out=2 /NOSYNC  ! out_buffers=2, in_buffers=default
SET ACQ /SYNC/MAIL     ! out_buffers=default, in_buffers=default
```

An **invalid** command would be

```
SET ACQ /MAIL/NET      ! illegal exclusive qualifiers
```

6.3.3 GOOSY Menu

GOOSY menus are specific to GOOSY components (see 6.9.3). The command `MENU` enters the GOOSY component menu. Some keys are defined for a shorter menu access: As input to a GOOSY component prompt, press $\boxed{\texttt{NEXT SCREEN}}$. Example:

```
$ MDBM                  ! start Data Base Manager
SUC: DBM> <NEXT SCREEN>  ! pressing this key enters the menu.
```

There are two types of menus, one for commands and one for the command parameters which is entered when you reach a full command (commands may be composed of several keywords like in DCL). The command menu looks like

```
keyword
Subcommands available ==================================================
keyword   * :  * :list of available subcommands
keyword     : short description
************** End of list ******************* End of list ***
```

The first line type is for a command which needs more subcommands, the second is the layout of an executable command. Entering a keyword of the first type, the next menu command level is displayed. Entering a keyword of the second type, the parameter menu of this command is displayed. Several keywords may be entered at once. The parameter menu looks like:

```
keyword keyword keyword
short description
Positional parameter list ====================================
parameter name |type|short description      :default
parameter name |type|short description      =default
Qualifier list ----------------------------------------------
qualifier      |SWI |short description      :default
Qualifier set list ------------------------------------------
set name       |SET|list of possible values  :default
********* End of list ****************** End of list ********
```

In this menu one moves the cursor around using the arrow keys and can overwrite the displayed default values. Note that qualifiers always must be preceeded by a slash (`/qualifier`). An $\boxed{=}$ sign in front of the default marks a required parameter. Some more keys on the separate keypad are activated as shown in figure 6.1 on page 80

Press the $\boxed{\text{KP\_PF2}}$ key to get a keypad layout. Press the $\boxed{\text{HELP}}$ key or $\boxed{\text{KP\_PF1}}$ and $\boxed{\text{KP\_PF2}}$ to get help information for the present level. At command level the following keys are activated:

```
<HELP> or <KP_PF1><KP_PF2> ! enter HELP for present level
key<RET>                   ! enter next level
<RET> or <KP_PF4>          ! return to previous level
<KP_4>                     ! Scroll to top (courtesy IBM)
<KP_5>                     ! Scroll to bottom  (courtesy IBM)
<KP_PF3>                   ! prompt for command line, return to menu
                           ! after execution
```

At command parameter level the following keys are activated:

```
<HELP> or <KP_PF1><KP_PF2> ! enter HELP for present level
```

Ctrl Z: Leave GOOSY menu

GOLD PF1	sho kpad HELP PF2	Enter command PF3	Break command PF4
— KP7	— KP8	— KP9	delete line -
text ⇑ KP4	text ⇓ KP5	— KP6	delete character ,
— KP1	— KP2	— KP3	— ENTER
— KP0		— .	

GOOSY command prompter

Ctrl A: toggle insert/overstrike mode
Ctrl E: move cursor to end of line
Ctrl H: move cursor to begin of line
Ctrl Z: Leave GOOSY Prompter

GOLD PF1	sho kpad HELP PF2	Enter command PF3	Break command PF4
— KP7	— KP8	— KP9	delete line -
text ⇑ KP4	text ⇓ KP5	— KP6	delete character ,
text ⇒ KP1	text ⇐ KP2	— KP3	execute command ENTER
next line KP0		— .	

GOOSY parameter prompter

Figure 6.1: The Special Keypad Layout for the GOOSY command (left) and parameter (right) menu.

```
<RET> or <KP_ENTER>        ! execute current command
<KP_PF4>                   ! return to previous level
<KP_4>                     ! Scroll to top (courtesy IBM)
<KP_5>                     ! Scroll to bottom  (courtesy IBM)
<KP_0> or arrow down       ! line down
arrow up                   ! line up
<DEL>                      ! delete character left of cursor
<KP_COMMA>                 ! delete character of cursor
<KP_1>                     ! shift text right (courtesy IBM)
<KP_5>                     ! shift text left (courtesy IBM)
<KP_MINUS>                 ! delete from cursor to end of line
<KP_PF3>                   ! prompt for command line, return to menu
                           ! after execution
<CTRL>A                    ! toggle between insert/overstrike mode
```

All defaults presently active are displayed in the menu. If the command is executed, the actual command line is displayed on top of the screen. You leave the component or prompter or menu by (several) $\boxed{\texttt{CTRL}}$Z.

6.3.4 ALIAS Names

GOOSY commands including parameter specifications can be replaced by alias names. These names are defined on two levels: global and environment. The environment level names are searched first. They are activated by the `CRENVIR` command and deactivated by the `DLENVIR` command. If no environment is active, only the global alias names are valid. Alias names cannot be abbreviated. They are implemented on GOOSY command level and DCL command level (DCL symbols). All alias names are deleted during logout. Therefore it is recommended to create alias names in the LOGIN.COM procedure using the DCL command `ALIAS CREATE`.

DCL Commands

DCL commands to handle alias names:

```
$ ALIAS CREATE name string environment /GLOBAL
$ ALIAS DELETE name environment /GLOBAL
$ ALIAS SHOW name environment /GLOBAL/ACTIVE
```

To create global alias names, omit the environment or use the `/GLOBAL` qualifier. Examples:

```
$ ALIAS CRE ANA "CREATE PROC MGOOANL"      !global alias ANA is created
$ ALIAS CRE INAC "INI ACQ /J11 NODE=" SUSI !envir.alias INAC is created
$ ALIAS SHO ENV=SUSI                       !envir.alias names of SUSI
$ ALIAS SHO                                !All active alias names
$ ALIAS SHO /GLOB                          !Global alias names only
$ ALIAS SHO INAC                           !Alias INAC
```

GOOSY Commands

Sometimes it is useful to create alias names from GOOSY command level. GOOSY commands to create, delete and show alias names are implemented in the GOOSY prompter, in the standalone Data Base Manager and the stand alone display. The arguments are the same as shown above, but the commands begin with the verb to fit into the GOOSY commands.:

```
DBM> CREATE ALIAS name "string"
GOOSY> SHOW ALIAS
MDISP> DELETE ALIAS name
```

Alias names created by GOOSY commands are not defined as DCL symbols.

6.3.5 User Defined Commands

Besides the commands provided by GOOSY the user may create his own commands. The command is created by routine `C$CRECM` which declares the parameter list and specifies the action routine to be called by the command. Routine `C$DSPMN` is called to prompt for command input. An example can be found in the file GOO$CMD: MCMD.PPL.

6.4 Data Management

A Data Base is located in a file and has a Data Base name. It is strongly recommended to use the same name for the file and the Data Base. The file type should be .SEC. A logical name may be created for the Data Base name. To activate a Data Base it must be **mounted**. It is dismounted during a system shutdown or by command. If a Data Base runs out of space, it can presently NOT be expanded.

The data region of a Data Base is splitted into **Pools**. All Data Elements are stored in Pools. A Pool (and all Data Elements in the Pool) can be accessed by a program with READ ONLY protection or with READ/WRITE protection. Pools must be created. They are automatically expanded if necessary, up to the space available in a Data Base. Similar to a OpenVMS disk, Data Elements of a GOOSY Data Base are organized in **Directories**. The user must create Directories to use them. A diagram of the simplified Data Base structure is shown in figure 7.1 on page 149.

Data Elements can be of atomic Types (scalars or arrays), or of structure Type (PL/1 structures). Besides the data structure a Data Element can be indexed (one or two dimensional). Such Data Elements are called name arrays. Each name array member has its own data and Directory entry. Similar to PL/1, the variables in a structure are called members. All GOOSY Data Elements like conditions, spectra, pictures, Dynamic Lists, etc. are kept in Data Bases.

Normally one Data Base is adequate for one analysis. The Data Base and its Data Elements are created by commands. **Presently all Data Elements must be created before starting an analysis**. A DCL command procedure should be written to do this. An example of such a DCL command procedure can be copied from the file GOO$EXAMPLES:DB.COM and adapted to the needs of the experiment. Presently at least one condition and one spectrum must be created in a Data Base used for analysis.

6.4.1 Create Data Bases

A new Data Base can be created and preformatted by the DCL command:

```
$ CREDB ?
$ CREDB basename filename size[KB]
       /SPECTRA=s    ! maximum number of spectra
       /PICTURE=p     ! maximum number of picture frames
       /CONDITIONS=c  ! maximum number of conditions
       /DIRECTORIES=d ! maximum number of Directories
       /POOLS=p       ! maximum number of Pools
       /POLYGON=p     ! maximum number of polygons
       /DYNLIST=d     ! maximum number of Dynamic Lists
```

The quotation mark enters a little menu. This command creates the Directories $SPECTRUM, $CONDITION, $PICTURE, $POLYGON, $DYNAMIC, DATA and the Pools $SPEC_POOL, $COND_POOL, $PIC_POOL, $DYNAMIC and DATA. The command procedure creating the

GOOSY Data Base can be saved by qualifier `/SAVE=file`. This file may be edited and used later instead of `CREDB`. For additional information for this command use DCL `HELP CREDB`. An example is shown in section 6.14 on page 132. Data Bases are accessible on one node by all programs started by the same OpenVMS user on that node.

6.4.2 Mounting and Dismounting Data Bases

Before accessing a Data Base, it must be mounted (already done by `CREDB`). This is done by issuing one of the DCL commands:

```
$ MDBM MOUNT BASE basename filename
$ MOBASE basename filename
```

GOOSY should then tell you that the Data Base `basename` has been mounted as a global section and is opened and ready. `filename` is the Data Base file name (a VAX/AXP-OpenVMS global section file). It is strongly recommended that the Data Base name is the same as the file name and that the filename has the type .SEC. The Data Base remains mounted until the node is rebooted or the Data Base is dismounted explicitely by one of the DCL commands:

```
$ MDBM DISMOUNT BASE basename
$ DMBASE basename
```

NOTE, however, that the Data Base remains mounted as long as there are programs active using it, even if the message says that there is 'no such (global) section'.

Only processes running on the same node of a cluster are allowed to share Data Bases. If you need a Data Base already mounted on a different node, change either to that node or dismount the Data Base on the other node and mount it on your node. A list of mounted Data Bases is displayed by command

```
$ SSEC
```

One can define logical names for Data Base names. They should be defined with DEFINE/JOB to be known for all subprocesses.

For a detailed description refer to the manual 'GOOSY Data Management'.

6.4.3 Compressing and Decompressing Data Bases

If a Data Base is not used for some time, it can be compressed. The dismounted and compressed Data Base requires in most cases much less disk space. It must be decompressed before it can be mounted again.

```
$ MUTIL COMPRESS BASE basename file
$ MUTIL DECOMPRESS BASE file basename basefile
```

'Basename' and 'basefile' are the names of the Data Base and Data Base file. 'File' is the name of the file for the compressed Data Base. The default file type is .CSEC. The base must be mounted to be compressed.

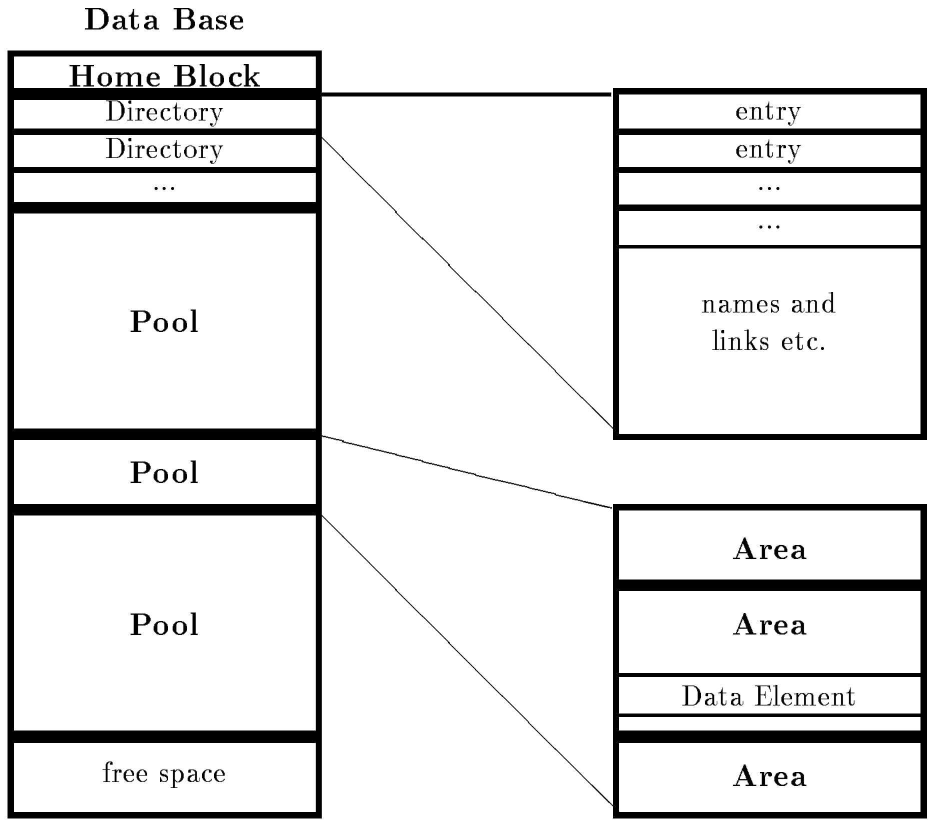


Figure 6.2: The simplified structure of a GOOSY Data Base.

6.4.4 Copying and Expanding Data Bases

A data base can be expanded only when it is copied. To copy a mounted base and to expand it to a bigger size, use

```
    $ MUTIL COPY BASE basename file SIZE=size
```

'Basename' is the name of the Data Base. 'File' is the name of the file for the new Data Base. The default file type is .SEC. The base must be mounted to be copied.

6.4.5 Save Data Bases

Data Bases are global sections. If they are in use by any program, parts of the Data Base are kept in the memory. When, due to a powerfail, the VAX crashes, the parts of the Data Base in the memory cannot be saved into the disk file. To force a dump of the Data Base from memory into the disk file one can use the command `UPDATE BASE`:

```
$ MDBM UPDATE BASE basename
```

6.5 GOOSY Data Elements

As mentioned above, all data is located in Data Bases. Data Bases are organized in Directories, similar to OpenVMS. All Data Elements have names which are inserted in a Directory. Various commands are provided to handle any Type of Data Elements, as CREATE, DELETE, SHOW, COPY, SET. In commands Data Element specifications have the general form:

```
node::base:[directory]name(index).member(index)
```

The node name is used in the future for remote Data Base access. It is presently not used. An asterisk or empty string denotes the current (local) node. All Data Elements may be structured like in PL/1. In addition - and in contrast to a file system - Data Elements may be composed of an array of Data Elements. We call such an array **name array** to distinguish it from an array inside the data. E.g. a spectrum consists of some header information and a data array, but the spectrum name may be indexed. Each member of such a name array keeps the same information as a single spectrum. The smallest referable entity of a Data Element is called a **member**.

There are some GOOSY Data Elements which are handled by special commands. These are described in the next sections.

6.5.1 Conditions

By default, conditions are kept in the Directory $CONDITION. GOOSY conditions are independent of spectra or coordinates (parameters), as opposed to SATAN analyzers. All kinds of conditions are executed as specified in a Dynamic List (see below). They may then be used as filters for spectrum accumulation and/or scatter plot. In an analysis routine they are executed by the macro $COND . Each condition has TRUE and FALSE counters and freeze, result, and preset bits. The following different condition types are implemented:

Window Conditions

A window condition keeps n window limits. Up to eight may be used in a Dynamic List. Each limit pair may be applied to a different object. An object may be any member of a Data Element which is a BIN FLOAT(24), BIN FIXED(31) or BIN FIXED(15) number. The condition is TRUE if all subwindows are TRUE.

Multiwindow Conditions

The difference to normal window conditions is that there is one result bit for each subwindow. In a Dynamic List any number of subwindows is processed. All subwindows are applied to the same object. The result bits can be used as filters for spectrum array accumulations. The number of the last TRUE subwindow may be used to select a spectrum array member for accumulation (See MULTIWINDOW and INDEXEDSPECTRUM in Dynamic Lists).

Pattern Conditions

Similar to the windows, the pattern conditions may keep n subpatterns. Up to eight may be checked in a Dynamic List. Each subpattern is compared to a different object which can be any Data Element member of Type BIT(16) or BIT(32) ALIGNED. The condition is TRUE if all subpatterns match. There are four matching modes:

1. IDENT
   Pattern and object must be identical.
2. ANY
   Pattern and object must have at least one common bit set.
3. INCL
   TRUE if all bits set in the pattern are set in the object (like IDENT inclusive additional bits set only in the object).
4. EXCL
   TRUE if all bits set in the object are set in the pattern (like ANY exclusive additional bits set only in the object).

In addition single bits in the objects can be inverted before testing.

Function Conditions

The user may write special routines for more complex conditions. These routines must be linked in a sharable image and can then be dynamically loaded. In the Dynamic List any members of Data Elements may be specified as arguments for these routines. The first argument, however, must be a BIT(8) ALIGNED returning the result.

Polygon Conditions

A polygon is created and modified independent of polygon conditions. Therefore several polygon conditions may reference the same polygon, but with different objects (coordinates). The polygon and objects are bound to the condition either by creation or by inserting in the Dynamic List. The execution time is similar to window conditions.

Composed Conditions

This may be any boolean expression of other conditions.

6.5.2 Polygons

By default, polygons are kept in the Directory $POLYGON. Polygons may be created, displayed, modified, copied and deleted. They can be specified by graphic input or numerically. They are used by one or more conditions.

6.5.3 Spectra

By default, spectra are kept in the Directory $SPECTRUM. A spectrum is composed of several Data Elements. The user need not be concerned with that, but in a `SHOW DIRECTORY` command these Data Elements will be listed. Spectra may be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24). The dimensionality can be up to two. Spectra may be filled in a Dynamic List Entry or by macro $ACCU .

Spectra are created as digital or analog spectra.

- Digital spectra are used to accumulate integer or bit variables. The integer binsize specifies the number of input bins to be incremented in one spectrum bin. Bit spectra should be dimensioned (1,16) or (1,32), respectively, with binsize 1.
- Analog spectra are used to accumulate float variables. The binsize specifies an interval. The lower limit of the interval is inclusive, the upper limit exclusive. Therefore the upper spectrum limit is exclusive.

GOOSY Spectra can be extracted from a Data Base and sent to the IBM to be processed by SATAN. This is done by:

```
$ MDBM DUMP SPECTRUM name OUTPUT=file
$ SIBMSPEC file VSAMlib /RUNID=id
```

The spectra dumped into 'file' are inserted into a SATAN VSAM library.

6.5.4 Calibrations

By default, calibrations are kept in the Directory $CALIB. Similar to spectra calibrations are sets of several Data Elements. They keep a calibration table which is used to calibrate the spectra data when displaying them. Each calibration can be connected to an arbitrary number of spectra.

6.5.5 Pictures

By default, pictures are kept in the Directory $PICTURE. Similar to spectra pictures are sets of several Data Elements. They keep information to display several frames containing spectra or scatterplots. Up to 64 frames may be displayed on one screen.

6.5.6 User Defined Data Elements

Besides the GOOSY Data Elements the user may define and create his own Data Elements. This may be done by GOOSY commands or by subroutine calls in a program. The following steps must be performed:

1. Put the PL/1 source declaration of the Data Element in a text library. The name of the structure should be used as the name for the library module. The declaration must declare a based structure. A base pointer may be specified (should be STATIC).

2. Create a Directory in a Data Base (optional)
3. Create a Pool in a Data Base (optional)
4. Create the Data Element Type, using the new PL/1 structure in the text library.
5. Create the Data Element of the new Type.

If the declaration contains REFER members, the Data Element can be created only in a program, because the REFER values must be specified. To access the Data Element in a program, include the library module declaring its structure and call $LOC macro to receive the pointer to the Data Element. An example is shown in section 6.14 on page 132.

6.5.7 Buffer and Event Types

GOOSY listmode data files contain buffers. The buffer and event headers are SA$bufhe and SA$evhe, respectively. The declarations can be found in the text library GOOINC. The third and fourth word in a buffer or event header specify the type and subtype of the buffer or event, respectively. Presently the following buffer types are defined. The numbers are defined in the text library GOOINC($BUFREP). Note the difference of the event structure in a buffer or in a data base.

1. **Buffer type=2, subtype=1, event type=1, subtype=1**
   Buffers formerly used by the J11 based single CAMAC crate system. The J11's write now buffers of type 4, subtype 1 (see below). There is an unpack routine A$UPJ11 for these buffers. The structure of the event data element is SA$e1_1, which is found in the text library GOOTYP. These buffers are presently 8192 bytes long.

2. **Buffer type=3, subtype=1,2, event type=3, subtype=1,2**
   Buffers from analysis output in /COMPRESS mode. On output events are compressed by A$PACMP. The subtype specifies the compression mode. On input the events are expanded by A$UPCMP. The structure of the event data element is free. The pack routine stores the full data element into the output buffer behind an event header. On input the unpack routine copies the event without header from the buffer to the event data element.

3. **Buffer type=4, subtype=1, event type=4, subtype=1,2**
   This type is used by the J11 single crate system and by analysis output in /COPY mode. The event data element structure is SA$EVENT, which is found in text library GOOTYP. The subtype 1 stands for uncompressed events, subtype 2 for zero suppressed events. Analysis output event data elements are copied by A$PAEVT into the output buffer. These events are unpacked during input by A$UPEVT. The structure of the event data element is free, but must have a standard event header. These buffers are presently 8192 bytes long.

4. **Buffer type=6, subtype=1, event type=6, subtype=1**
   This type is used by the standard MBD system. The event data element structure is

SA$MBD, which is found in text library GOOTYP. These buffers are presently 8192 bytes long.

5. **Buffer type=10, subtype=1, event type=10, subtype=1,2,2..**
   This type is used by the standard VME system. The event data element structure is SA$VE10_1, which is found in text library GOOVME. The events are composed by subevents (Structure SA$VES10_1 in GOOVME). CAMAC and Fastbus structures are provided. These buffers are presently 16384 bytes long.

In the `START INPUT MAIL` command the appropriate buffer size must be specified! The default of 8192 matches the current default systems.

6.6 Data Base Manager

As an example for GOOSY commands and to get familiar with Data Elements, we will describe in more detail the Data Base Manager. It is invoced stand alone by the DCL command:

```
   $ MDBM                     ! start DBM
   SUC:DBM> <NEXT SCREEN>     ! pressing this key enters the menu.
```

A Data Base should have been created already, e.g. by CREDB. The first menu level looks like:

```
Subcommands available =====================================================
$ *          :  * :ATTACH,CALL,DCL,DEBUG,DEFINE,DIRECTORY,EXECUTE,EXIT,..
CALCULATE * :  * :SPECTRUM
CALIBRATE * :  * :SPECTRUM
CLEAR *      :  * :CAMAC,CONDITION,ELEMENT,PICTURE,SPECTRUM
COPY *       :  * :CONDITION,ELEMENT,MEMBER,POLYGON,SPECTRUM
CREATE *     :  * :ALIAS,AREA,BASE,CALIBRATION,CONDITION,DIRECTORY,...
DECALIBRATE :  * :SPECTRUM
DEFINE       :  * :PICTURE
DELETE *     :  * :ALIAS,CONDITION,ELEMENT,LINK,OVERLAY,POOL,..
DISMOUNT *   :  * :BASE
DUMP *       :  * :SPECTRUM
FREEZE *     :  * :CONDITION,SPECTRUM
HELP         : Access VMS HELP facility
LOCATE *     :  * :BASE,DIRECTORY,ELEMENT,ID,POOL,QUEUEELEMENT,TYPE
MENU         : Enter menu
MODIFY *     :  * :DIRECTORY,FRAME,TABLE
MOUNT *      :  * :BASE
READ *       :  * :CAMAC
SET *        :  * :CONDITION,LETTERING,MEMBER,SPECTRUM
SHOW *       :  * :ALIAS,AREA,CALIBRATION,CAMAC,CONDITION,DIRECTORY,..
START *      :  * :MR2000
STOP *       :  * :MR2000
SUMUP *      :  * :SPECTRUM
UNFREEZE *   :  * :CONDITION,SPECTRUM
UPDATE *     :  * :BASE,DYNAMIC
WRITE *      :  * :CAMAC
************** End of list ******************* End of list ***
Command:
Help: Help, PF2: Keypad, PF3: Enter command, PF4: Break, ENTER: Prev.menu
Subcommand :
```

One should work with the menu to get familiar with it. If you execute a command, the full command line will be displayed on top of the screen.

In the following we show some often used commands and their most important arguments. Examples can be found in section 6.14 on page 132. The commands can be given to the `DBM>` prompt or to the `GOOSY>` prompt if an environment with $DBM component is created. Note that in the following descriptions lower case names have to be replaced by meaningfull values. Uppercase names are keywords.

6.6.1 CREATE Commands

Create Directories

For creating Directories one should know that each name array member takes one entry in the Directory. Some GOOSY Data Elements take more than one entry, i.e. spectra four, conditions two, composed conditions three, and pictures one per picture plus one per frame.

```
CRE DIRECTORY directory 100 base       ! 100 entries
```

Create Pools

All Data Elements are allocated in Pools. Normally the default Pools created by command `CREDB` are adequate. One may, however, create additional private Pools. The poolsize is not a limit of the Pool, because it is extended automatically. One should at least specify the size of the largest Data Element to be allocated in the Pool.

```
CRE POOL pool 8192 base                ! size 8192 bytes
```

Create Data Element Types

To create a Data Element, one must specify the structure declaration. This is done by a PL/1 structure declaration. This declaration must be in a file or text library module. The name of the file or library module must be the name of the structure, respectively. It must be made known to the Data Base. This is done by `CREATE TYPE`:

```
CRE TYPE @library(module) base         ! Declaration from library
CRE TYPE @filename base                ! Declaration from file
```

Create Data Elements

```
CRE ELEMENT [directory]name pool type  ! Pool, Type, and dir. must exist
```

Create Polygons

Each polygon takes two entries in Directory $POLYGON. The default Pool is $PIC_POOL.

```
 CRE POLYGON name points
```

The number of points of the polygon are extended automatically if needed. The polygon can be modified by the `REPLACE POLYGON` command:

```
    REPLACE POLY name X=(x1,x2,x3,...) Y=(y1,y2,y3,...)
    REPLACE POLY name frame /MODIFY
```

In the second command the polygon is displayed in the specified frame and can be edited by cursor input. Using the `/DELETE` qualifier instead of `/MODIFY` existing points are deleted first. The polygon can be displayed by command `DISPLAY POLYGON`.

Create Conditions

Each condition takes two entries in Directory $CONDITION, except composed conditions which take three. The default Pool is $COND_POOL

```
   CRE COND WINDOW c (1,1000) 1          ! 1 subwindow
   CRE COND WINDOW c (1,1000) 2          ! 2 subwindows
                                         ! both (1,1000)
   CRE COND WINDOW c(10) (1,100)         ! 10 cond., 1 subw.
   CRE COND WINDOW c (1,100,1,200)       ! 2 subwindows
   CRE COND MULTI  c (1,1000) 100        ! 100 subwindows
   CRE COND PATTERN c '1'B               ! 1 subpattern
                                         ! padded right with 0
   CRE COND PATTERN c '1'B 2             ! 2 subpatterns
   CRE COND PATTERN c '1'B INV='1'B /ANY ! invert first bit
   CRE COND PATTERN c '11111'B /IDENT    ! identical match
   CRE COND COMP c "a | (x & y)"         ! a, x, y must exist
   CRE COND POLYGON c poly               ! polygon poly must exist
```

Window and pattern conditions are set by commands `SET CONDITION PATTERN` and `SET CONDITION WINDOW`, respectively. Window conditions can be set by cursor input with command `REPLACE CONDITION WINDOW`.

Create Spectra

Each spectrum takes four entries in Directory $SPECTRUM. Default Pool is $SPEC_POOL.

```
   CRE SPEC s L (0,1023) 10 /DIGITAL      ! BIN FIXED(31), binsize=10
   CRE SPEC s R (0,1023,0,255) (10,10)    ! BIN FLOAT(24), 2-dim.
   CRE SPEC s(10) L (-10,15) 0.1 /ANALOG  ! name array, binsize 0.1
```

Create Dynamic Lists

Each Dynamic List takes two entries in Directory $DYNAMIC. The default Pool is $DYNAMIC.

```
   CRE DYNAMIC LIST list 100     ! Dynamic List for 100 entries
```

Create Dynamic Entries

For Dynamic List Entries the objects for spectrum accumulation and condition checks, the spectrum increment and the index must be members of GOOSY Data Elements created already in the Data Base. Assume we have created a Data Element like this:

```
DCL P_SX$evt POINTER;
DCL 1 SX$evt BASED(P_SX$evt),
    2 patt     BIT(32) ALIGNED,
    2 geli(10) BIN FIXED(31),
    2 naj(10   BIN FIXED(15);
```

The structure should be declared BASED for references of the Data Element in PL/1 programs. This declaration is in our library TPRIV in module SX$EVT. We refer in the following examples to Data Element EVT of the above Type. We assume that conditions c,w,a(1:10) and spectra s and s2 already exist.

```
CRE TYPE @tpriv(SX$evt)                  ! Declaration in library
CRE ELEMENT [data]evt evtdata SX$evt     ! Directory DATA and
                                         !  Pool EVTDATA must exist

CRE DYNAMIC LIST list ENTRIES=100      ! Dynamic List for 100 Entries
CRE DYN ENTRY PATTERN list c PARAMETER=evt.patt
CRE DYN ENTRY WINDOW  list w PARAMETER=evt.geli(3)
CRE DYN ENTRY WINDOW  list a(1:10) PARA=evt.geli(1:10)
                                         ! condition name array!
CRE DYN ENTRY SPECTRUM list s PARAMETER=[data]evt.naj(1)
CRE DYN ENTRY SPECTRUM list s2 -
              PARAMETER=([data]evt.naj(1),[data]evt.geli(1)) -
              CONDITION=c
                                         ! 2-dim. spectrum
```

Create Pictures

Pictures take one entry in the $PICTURE Directory. Each frame takes one more entry. The default Pool is $PIC_POOL. First, a picture is created. Then the frames are specified.

```
CRE PICTURE pict 6 /NOPROMPT             ! 6 frames
MOD FRAME SCATTER pict 1 [data]evt.geli(1) [data]evt.naj(1)
                                         ! frame one: scatter
MOD FRAME SCATTER pict 2 [data]evt.geli(2) [data]evt.naj(2)
                                         ! frame two: scatter
MOD FRAME SPECTRUM pict 3 [$SPECTRUM]s
                                         ! frame three: spectrum
```

Create Calibrations

Each calibration takes two entries in Directory $CALIB. One for the main Data Element and one entry for the calibration table contents. First a calibration is created. Then it is connected to several spectra (for detail description see page 126):

```
CREATE CALIBRATION FIXED cal_1 1024    ! a calibration table with
                                       ! 1024 entries and fixed
                                       ! stepwidth between the
                                       ! uncalibrated values is
                                       ! created
CALIBRATE SPECTRUM s cal_1             ! spectrum "s" is calibrated
                                       ! with "cal_1"
```

6.6.2 SHOW Commands

The output of most SHOW commands can be stopped by $\boxed{\texttt{CTRL}}$Z.

```
SHOW CONDITION *                        ! list of all conditions
SHOW CONDITION c /FULL                  ! full information
SHOW CONDITION * /WINDOW                ! window cond. only
SHOW SPECTRUM *                         ! list of all spectra
SHOW SPECTRUM * /FULL                   ! full information
SHOW SPECTRUM s /DATA                   ! spectrum data
SHOW DYNAMIC LIST list *                ! all entries
SHOW DYNAMIC LIST list SPECTRUM         ! all spectrum entries
SHOW PICTURE *                          ! list of all pictures
SHOW PICTURE pict /FULL                 ! full information
SHOW ELEMENT [data]*                    ! list of Elements in [data]
SHOW ELEMENT [data]EVT /DAT             ! contents of [data]EVT
SHOW TABLE                              ! Show all counters of
                                        ! spectra and conditions
SHOW MEMBER [data]EVT.GELI(1)           ! show contents
```

6.6.3 CLEAR Commands

```
CLEAR SPECTRUM *                        ! Clear all spectra
CLEAR SPECTRUM s                        ! Clear spectrum s
CLEAR SPECTRUM s*                       ! Clear all spectra s*
CLEAR CONDITION COUNTER *               ! Clear all test/true counters
```

6.6.4 DELETE commands

```
DELETE SPECTRUM
DELETE CONDITION
DELETE DYNAMIC ENTRY
DELETE PICTURE
DELETE POLYGON
DELETE ELEMENT
```

Data Elements which are in use by any program cannot be deleted, i.e. the analysis program protects all Data Elements it references. The `DETACH ANALYSIS` command releases this protection.

6.6.5 Miscellaneous Commands

```
COPY SPECTRUM
COPY ELEMENT
COPY MEMBER
CALCULATE SPECTRUM
MODIFY DIRECTORY
SET MEMBER
```

6.7 Data Base Access

Besides the methods described in section 6.9.2 on page 102 it is easy to write a program or routine to access data elements in a data base. First one has to attach the data base. This is done by macro `$ATTACH`:

```
$ATTACH(BASE,base,W);
```

Then each data element to be accessed must be located by macro `$LOC`. This macro returns the pointer to the data element. The structure declarations of GOOSY data elements are included by

```
@INCLUDE $MACRO($MACRO);
```

The data base can be detached by

```
$DETACH(BASE,base);
```

An example can be found on GOO$EXAMPLES:TBASE.PPL.

6.8 Utility Commands

Some commands are executed in a separate program called MUTIL.

```
   $ MUTIL                      ! start utility program
   SUC:DBM> <NEXT SCREEN>       ! pressing this key enters the menu.
```

The first menu level looks like:

```
Subcommands available ==================================================
$ *             :  * :ATTACH,CALL,DCL,DEBUG,DEFINE,DIRECTORY,EXECUTE,EXIT,...
COMPRESS *      :  * :BASE
COPY *          :  * :BASE,FILE
CREATE *        :  * :ALIAS,PROGRAM
DECOMPRESS *    :  * :BASE
DELETE *        :  * :ALIAS
HELP            : Access VMS HELP facility
MENU            : Enter a menu driven command dispatcher
SET *           :  * :LOCK
SHOW *          :  * :ALIAS,LOCKS
TYPE *          :  * :FILE
*************** End of list ***************** End of list *********
Command:
Help: Help, PF2: Keypad, PF3: Enter command, PF4: Break, ENTER: Prev.menu
Subcommand :
```

6.9 Starting GOOSY

6.9.1 Steps

To start GOOSY one must perform the following steps (See examples in section 6.14 on page 132):

1. Prepare the hardware and the online programs for the frontend processors. See the 'GOOSY Hardware Manual' for this purpose.
2. Create a Data Base and all Data Elements like spectra, conditions, pictures, Dynamic Lists, and Dynamic List Entries. This should be done in a DCL procedure. An example is found in the file GOO$EXAMPLES:DB.COM.
3. Prepare your analysis program. (see below).
4. Create an environment (see below).
5. Initialize the acquisition and activate Dynamic Lists, if needed.
6. Start data taking or analyzing.

These steps are described in more detail in the next sections.

6.9.2 Analysis Program

Event Loop

The event loop performs the following steps:

1. Clear execution bits.
2. Call unpack routine depending on buffer type.
3. Call user analysis routine (optional).
4. Call Dynamic List Executor (optional).
5. Fill output event into output buffer (optional).

Figure 23.1 on page 305 shows the steps performed in the event loop. First an event is copied from the input buffer to the Data Base. It may be expanded during this step. Then a user analysis routine is called (optionally) which may access the event in the Data Base and other Data Elements. It may write new values into a Data Element for output. Then the Dynamic List Executor is called. Then (optionally) the output Data Element is copied into the output buffer which is delivered to the output channels.

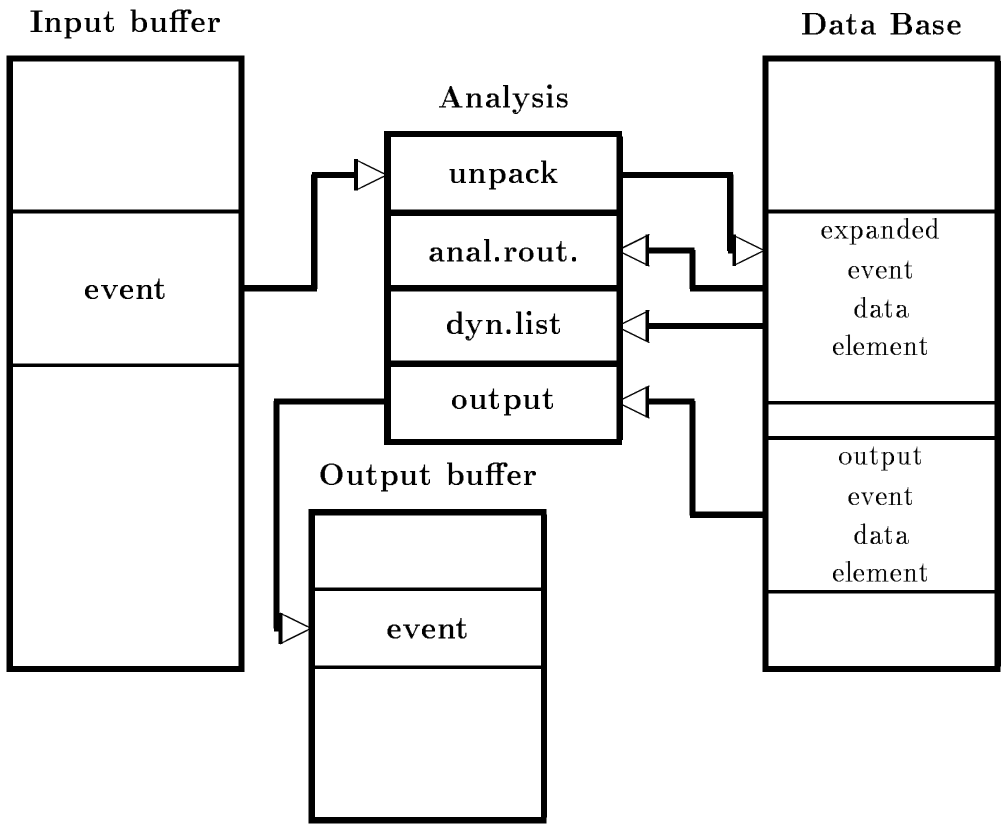


Figure 6.3: Moving events between I/O buffers and data bases in the event loop.

User Analysis Program

If you need an analysis routine, you must link your specific analysis program by

```
LANL anal            !  standard unpack
```

The object file 'anal' must contain an analysis routine called X$ANAL. The unpack routines for J11 generated data, standard MBD data, and analysis output data are provided and linked automatically. See HELP LANL for more information.

Standard Event Unpack Routines

For all standard GOOSY event types unpack routines are provided: Events written by the J11 based single CAMAC crate system are copied from buffers into a Data Element DB:[DATA]EVENT of Type SA$EVENT declared in the text library GOOTYP. If the events should be copied to another Data Element, it can be specified by the `SET EVENT INPUT` command.

Events written by the MBD controlled CAMAC system are copied from buffers into a Data Element DB:[DATA]EVENT of Type SA$MBD declared in the text library GOOTYP. If the events should be copied to another Data Element, it can be specified by the `SET EVENT INPUT` command.

Events written by an analysis are copied from buffers into a Data Element DB:[DATA]EVENT of any Type (Type must match the original event Data Element Type). If events should be copied to another Data Element, it can be specified by the `SET EVENT INPUT` command.

User Event Unpack Routine

User unpack routines can be loaded dynamically and are then called instead of the standard routines. Unpack routines get passed the pointer to a data buffer. They control by the return code what GOOSY will do after the call:

```
RETURN(1);               Process event.
RETURN(XIO_NOMOREEVENT); Provide a new buffer.
RETURN(XIO_SKIPEVENT);   Skip the event.
RETURN(XIO_NOOUTPUT);    Suppress the output of this event
                         (ignored, if output is not enabled).
```

The return codes must be declared by

```
@DCL_MSG(XIO_NOMOREEVENT);
@DCL_MSG(XIO_SKIPEVENT);
@DCL_MSG(XIO_NOOUTPUT);
```

The unpack routine copies one event from the buffer to a GOOSY Data Element which must exist in a Data Base. It must have an initialization entry. This entry is called during startup of the analysis program and must locate all Data Elements it needs. Use the $LOC macro for this purpose. One pointer and one Longword is passed. Both are zero for normal startup. The `SET EVENT INPUT` command calls the initialization entry and passes the pointer and length of the specified Data Element. The macros must be declared by

```
@INCLUDE $MACRO($MACRO);
```

A template is available from the file GOO$TEST:X$EVENT.PPL.

User Analysis Routine

Besides the dynamic analysis controlled by commands the user may write an analysis routine doing some calculations, checking conditions and accumulate spectra. This routine must be named **X$ANAL**. It is called without arguments. It must provide an entry **$XANAL** doing all necessary initializations. This entry is called during the startup of the analysis program.

Typically one calls the $LOC macros to locate conditions, spectra or Data Elements to be used in the routine. See the HELP for detailed description. In the X$ANAL routine one uses $ACCU and $COND macros to check conditions and accumulate spectra. The macros must be declared by

```
    @INCLUDE $MACRO($MACRO);
```

A template is available from the file GOO$TEST:X$ANAL.PPL. An example is described in section 6.14 on page 132 and is available from the file GOO$EXAMPLES:X$ANAL.PPL.

The execution of the routine can be disabled and enabled by command:

```
SET ANALYSIS /NOANAL    ! disable
SET ANALYSIS /ANAL      ! enable
```

Analysis Output

Sometimes it is useful to output list mode data from an analysis program. This output is started by the command:

```
START ANALYSIS OUTPUT
```

Then the Data Element DB:[DATA]NEWEVENT is packed into output buffers. Two packing modes are selected with the command qualifiers `/COMPRESSED` or `/COPY`. For `/COMPRESSED` mode the Type (structure) of DB:[DATA]NEWEVENT can be chosen freely by the user. For `/COPY` mode the Data Element must have an event header like GOOINC(SA$EVHE). The Data Element must exist in the Data Base. If the another data should be copied, it can be specified by the `SET EVENT OUTPUT` command. The buffers are written to DECnet (can be read by a second analysis on a different node) and optionally to a file. Output is stopped by the command:

```
STOP ANALYSIS OUTPUT
```

If another Data Element should be used for output, it can be specified by the command:

```
SET EVENT OUTPUT
```

Output is done event by event. If the analysis routine returns status XIO_NOOUTPUT, no output event is written.

Macros

The macro calls for analysis routines can be inserted by the LSEDIT editor using the $\boxed{\texttt{F\_8}}$ key.

$LOC macro to locate spectra, conditions and Data Elements. For a one dimensional name array use `$LOC1`, for a two dimensional `$LOC2` macro. You must include a check on the successful execution just behind each $LOC macro. The type argument must be I,L,R or S for spectra and WC,PC,MW or POLY for conditions. Examples (W means write access):

```
$LOC(SPEC,base,$SPECTRUM,spectr,W,type);/* locate spectrum in Data Base  */
$LOC(COND,base,$CONDITION,cond,W,type); /* locate condition in Data Base */
$LOC(DE,base,directory,name,W,type);    /* locate Data Element           */
   Declares and returns a pointer P$_base_dir_name.
   Declares and returns a length  L$_base_dir_name.
$LOC1(SPEC,base,$SPECTRUM,spectrum,1,5,W,t);/* locate spectrum array      */
$LOC1(COND,base,$CONDITION,cond,1,8,W,t);/* locate condition array       */
$LOC1(DE,base,directory,name,1,2,W,t);  /* locate Data Element array     */
   Declares and returns a pointer array P1$_base_dir_name.
   Declares and returns a length array L1$_base_dir_name.

IF ^STS$success THEN @RET(STS$value);   /* Check execution success       */
```

$DE macro to reference GOOSY Data Element members in a program.

```
X=$DE(base,directory,name,member);   /* Copy value to local var. X */
X=$DE(DB,data,EVENT,SA$EVENT.IA$EVENT(I));
```

The Data Element must be located by `$LOC` macro.

$COND macro to execute condition checks. This macro executes window, multiwindow and pattern conditions. Pattern conditions can be of Type `ANY, IDENT, EXCL or INCL`. For a one dimensional name array condition use `$COND1`, for two dimensional the `$COND2` macro. Examples:

```
$COND(WC,base,$CONDITION,cond,result,1,x1);             /* one subwindow  */
$COND(WC,base,$CONDITION,cond,result,2,x1,x2);          /* two subwindows */
$COND(ANY,base,$CONDITION,cond,result,1,x1);            /* one subpattern */
$COND(MW,base,$CONDITION,cond,result,1,x);              /* multi window   */
$COND(MWI,base,$CONDITION,cond,result,1,x);             /* multi window   */
$COND(POLY,base,$CONDITION,cond,result,2,x,y,poly);     /* polygon        */
```

In the following calls `index` is the condition name index.

```
$COND1(WC,base,$CONDITION,cond,index,result,1,x1);      /* one subwindow  */
$COND1(WC,base,$CONDITION,cond,index,result,2,x1,x2);   /* two subwindows */
$COND1(ANY,base,$CONDITION,cond,index,result,1,x1);     /* one subpattern */
```

$ACCU macro to accumulate one or two dimensional spectra. For a one dimensional name array spectrum use `$ACCU1`, for two dimensional the `$ACCU2` macro. Examples:

```
$ACCU(L,base,$SPECTRUM,spec,incr,1,x1);                 /* one dimension  */
$ACCU(R,base,$CONDITION,spec,incr,2,x1,x2);             /* two dimensions */
```

In the following calls `index` is the spectrum name index.

```
$ACCU1(L,base,$SPECTRUM,spec,index,incr,1,x1);        /* one dimension  */
$ACCU1(R,base,$CONDITION,spec,index,incr,2,x1,x2);    /* two dimensions */
```

$SPEC macro to fill one or two dimensional spectra. The value is copied into the spectrum channel. For a one dimensional name array spectrum use `$SPEC1`, for two dimensional the `$SPEC2` macro. Examples:

```
$SPEC(L,base,$SPECTRUM,spec,value,1,x1);               /* one dimension  */
$SPEC(R,base,$CONDITION,spec,value,2,x1,x2);           /* two dimensions */
```

In the following calls `index` is the spectrum name index.

```
$SPEC1(L,base,$SPECTRUM,spec,index,value,1,x1);        /* one dimension  */
$SPEC1(R,base,$CONDITION,spec,index,value,2,x1,x2);    /* two dimensions */
```

6.9.3 The GOOSY Environment

As shown above, the Data Base Manager and the Display Program may be executed standalone on DCL level or bundled together with other GOOSY programs in an environment. The Transport Manager and Analysis components can run ONLY in an environment. In an environment several components are started together. Figure 20.1 on page 254 shows the communication between environment components. Commands executed by environment components are dispatched by the GOOSY prompter from one terminal. Therefore, on each terminal one has to create an environment. Commands given from that terminal are executed by the components running in that environment. The Data Bases, however, are shared between environments. Therefore the display may run in a different environment than the analysis. To create an environment with optional components, use the DCL command `CRENVIR`:

```
$ CRENVIR ?
$ CRENVIR environment myanal /ONLINE
$ CRENVIR environment myanal /OFFLINE
$ CRENVIR environment /ONLINE/DEFAULT
$ CRENVIR environment /OFFLINE/DEFAULT
```

The quotation mark enters a little menu. The environment name must be unique within a user group on one VAX node. It can be one to four characters long. The difference between `/ONLINE` and `/OFFLINE` is that with `/ONLINE` the transport manager will be started. By the qualifier `/DEFAULT` the standard GOOSY analysis program will be used. Otherwise the program specified by 'myanal' (default is MGOOANL) will be started.
The analysis is started by default with priority 3. Specify another priority by `CRENVIR ... /PRIO=p`. Similar, the DCL command

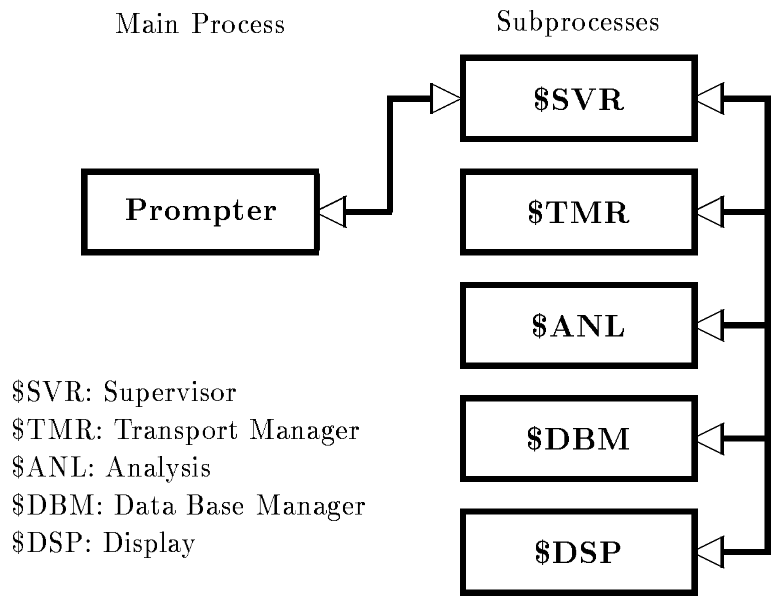


Figure 6.4: GOOSY components shown as VMS processes.

```
   $ DLENVIR
```

deletes the present environment including all components (subprocesses).

6.9.4 The GOOSY Prompter

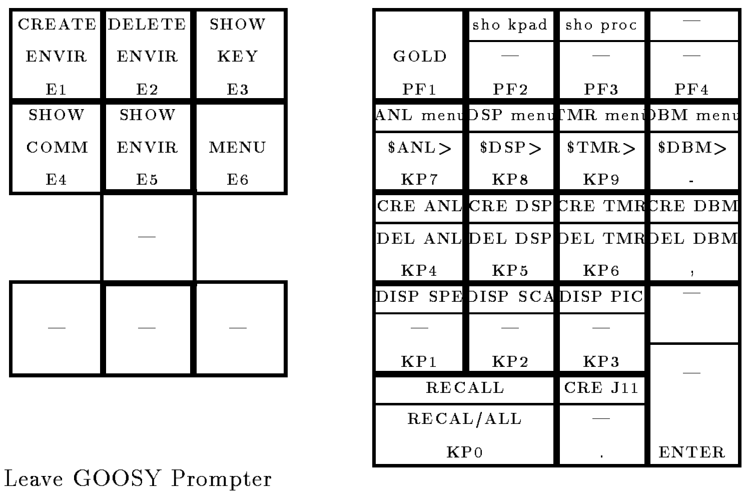
The GOOSY prompter can only be used after an environment was created! The GOOSY prompter is entered by the DCL command `GOOSY`:

```
   $ GOOSY
it prompts with
   SUC:GOOSY>
```

Now you can enter any GOOSY command. You leave the GOOSY prompter by typing `CTRL` Z. Single GOOSY commands can be given under DCL by a preceding `GOOSY` or just `G`. This allows to execute all GOOSY commands as DCL commands or in DCL command procedures which means also in batch jobs. Vice versa, a single DCL command can be executed by the GOOSY prompter:

```
   $ G SHOW TP KEYPAD              !Show keypad layout
```

The upper key values are the simple key hits, the lower are entered with a preceding GOLD(=PF1)-key hit.



Ctrl Z: Leave GOOSY Prompter

Figure 6.5: The Special Keypad Layout for the GOOSY prompter.

```
$ GOOSY                           !Enter GOOSY prompter
SUC:GOOSY> $ DCL "DCL-command"    !Execute DCL command
SUC:GOOSY> <Ctrl>Z                !Leave GOOSY prompter
$
```

Note, however, that "DCL-command" executes in a separate (spawned) temporary process. The GOOSY prompter menu is displayed by pressing the $\boxed{\texttt{NEXT SCREEN}}$ key of your terminal key board.

The GOOSY prompter interprets some special function keys. The definitions are made in GOO$EXE:INI_TP0.COM. A help setup is displayed by the $\boxed{\texttt{KP\_PF2}}$ key. Figure 20.2 on page 256 shows the keypad layout.

Now, entering the GOOSY prompter, the menus for the different components are entered by:

```
<KP_7>      ! $ANL : Analysis commands menu
<KP_8>      ! $DSP : Display commands menu
<KP_9>      ! $TMR : Transport Manager commands menu
<KP_MINUS> ! $DBM : Data Base Manager commands menu
```

Similar, single components can be deleted and created by special keys:

```
[<PF1>]<KP_4>     ! [delete]create $ANL : Analsysis
[<PF1>]<KP_5>     ! [delete]create $DSP : Display
[<PF1>]<KP_6>     ! [delete]create $TMR : Transport manager
[<PF1>]<KP_COMMA> ! [delete]create $DBM : Data Base Manager
```

TMR: Transport Manager Program
MBD: Microprogrammed Branch driver
J11: Auxiliary crate controller
VME: VME frontend system

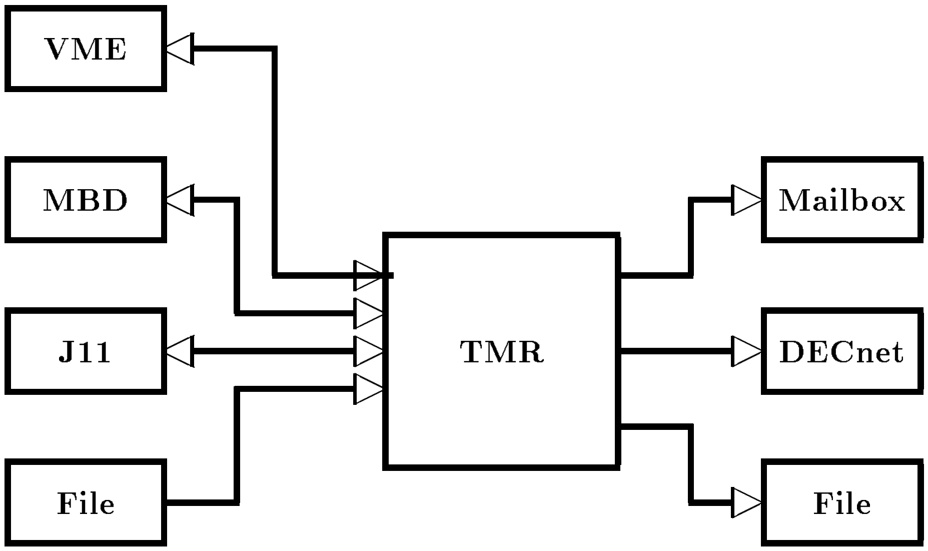


Figure 6.6: The input and output channels of the Transport Manager

Of cause, all GOOSY commands can be entered by line directly to the GOOSY prompter.

6.9.5 GOOSY Components

The following sections describe the GOOSY components running in an environment.

The Transport Manager

The Transport Manager $TMR is the interface to the frontend systems and/or disk or tape files and dispatches the data buffers. It may get input from an MBD, J11, CVC, VME system, mailbox or a file and may write buffers to disk, tape, mailbox, and/or DECnet. Actually the mailbox and DECnet are filled anytime they are read out. Writing to tape or disk is controlled by commands. There is a menu of Transport Manager commands available which can be activated by:

SUC: GOOSY> $\boxed{\texttt{KP\_9}}$

The menu is self-explanatory and contains short descriptions of the available commands.

The GOOSY PAW server

PAW clients may connect to this server. The server reads data from the Transport Manager through a mailbox, filters them and sends an event stream to the connected clients.

The Data Base Manager

The Data Base Manager $DBM executes all commands to maintain GOOSY Data Bases and Data Elements, i.e. Dynamic Lists, spectra, conditions etc.. The menu is activated by:

SUC: GOOSY> $\boxed{\texttt{KP\_MINUS}}$

The Data Base Manager may be started directly under DCL. In this case it is called by the DCL command

```
$ MDBM
```

`it answers with`

```
SUC: DBM>
```

Now the $\boxed{\texttt{NEXT SCREEN}}$ key will enter the menu.

The Display

The display menu is activated by:

SUC: GOOSY> $\boxed{\texttt{KP\_8}}$

You can start one more display in the same environment by the DCL command:

```
$ GOOSY CREATE PROCESS DISP name
```

where name is a 4 character name. However, if you control two displays from one terminal, you have to prefix all display commands by `name`> ($DSP is the default display):

SUC: GOOSY> `$DSP`> `DISPLAY PICTURE`
SUC: GOOSY> `name`> `DISPLAY PICTURE`

The process is started by default with priority 3. Specify another priority by `CREATE PROCESS ... PRIO=p`. The display may be started directly under DCL. In this case it is called by

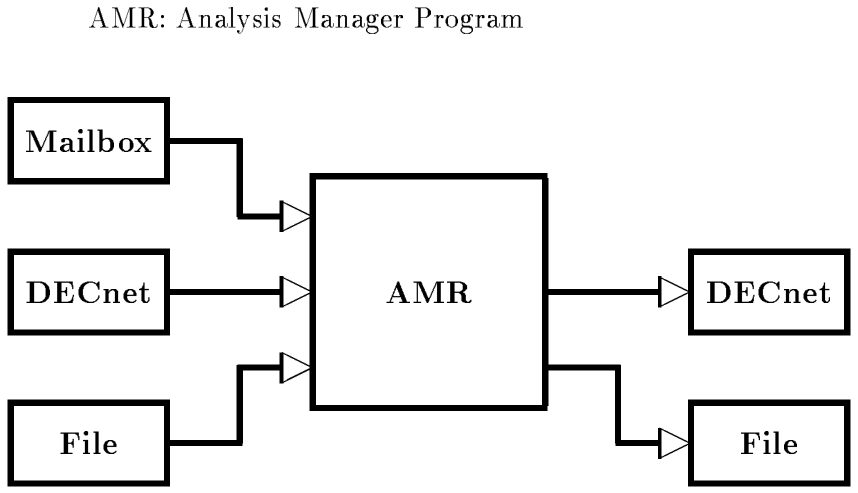


Figure 6.7: The input and output channels of the Analysis Manager (analysis program)

```
$ MDISP
```

it answers with

```
SUC: DISP>
```

Now the NEXT SCREEN key will enter the menu. However, no scatter plots can be displayed in this 'stand alone' mode.

The Analysis Program

The program MGOOANL is normally linked by the user with the DCL command `LANL`. It is started as $ANL component. The $ANL menu is activated by:

SUC: GOOSY> KP_7

For the correct function of the analysis routines one must make sure that the Data Base has already been mounted (so that $ANL understands the names of variables, their structures and where they reside). A default analysis program MGOOANL is provided on GOO$EXE. If GOOSY displays an error message saying that the Data Base could not be attached (or global section not found) then one has mount the Data Base and initialize the analysis by the command:

```
$ GOOSY MOUNT BASE base file
$ GOOSY INITIALIZE ANALYSIS
```

The standard analysis program for data input from a single CAMAC crate J11 system, an MDB system, or from analysis output is started by $\boxed{\texttt{KP\_PERIOD}}$ or the `/DEFAULT` qualifier of the `CRENVIR` command.

6.10 Dynamic Analysis

6.10.1 Activating Dynamic Lists

Dynamic Lists are Data Elements in a Data Base. Each condition check, spectrum accumulation, or scatter plot is an Entry in a Dynamic List. The creation of Dynamic Lists and Entries should be done in the DCL command procedure building the Data Base. Dynamic List Entries are executed per event and may be created and deleted dynamically (parallel to a running analysis).

Several Dynamic Lists may be executed in one analysis program. Dynamic List execution is activated by attaching to it.

```
SUC:GOOSY> ATT DYN LIST d1
```

There may be up to ten different lists attached at the same time. If it is desired to stop the execution of one list and to start the execution of another one, one would type e.g.:

```
SUC:GOOSY> DETACH DYN LIST d1
SUC:GOOSY> ATTACH DYN LIST d2
```

Already attached lists can also be stopped and started by

```
SUC:GOOSY> STOP DYN LIST d3
SUC:GOOSY> START DYN LIST d3
```

Note that the execution order is the order of attachment. This order may be changed with the `DETACH/ATTACH` commands, but not with the `STOP/START` commands. Furthermore, the `STOP/START` sequence is much faster. The following `SHOW` command gives you information about the Dynamic Lists actually executing:

```
SUC:GOOSY> SHO DYN ATT * * !Show all Dynamic Lists of all types
```

There is a "top" command to disable and enable Dynamic List execution at all:

```
SUC:GOOSY> SET ANALYSIS /NODYN   ! disable all Dynamic Lists
SUC:GOOSY> SET ANALYSIS /DYN     ! enable all Dynamic Lists
```

6.10.2 Related Commands

```
CREATE DYNAMIC LIST      listname
DELETE DYNAMIC LIST      listname
SHOW   DYNAMIC LIST      listname
CREATE DYNAMIC ENTRY type listname
DELETE DYNAMIC ENTRY type listname
ATTACH DYNAMIC LIST      listname (for Analysis program only)
DETACH DYNAMIC LIST      listname (for Analysis program only)
SHOW   DYNAMIC ATTACHED  listname (for Analysis program only)
STOP   DYNAMIC LIST      listname (for Analysis program only)
START  DYNAMIC LIST      listname (for Analysis program only)
```

The following switches apply for the `CREATE DYNAMIC ENTRY` commands:

`/UPDATE` The modification becomes active immediately (also for the `DELETE DYNAMIC ENTRY` command).

`/MASTER` Valid for conditions (except multiwindow) and procedures. Master Functions are executed first of all other Entries. Master conditions are executed first of all other conditions. If a master condition's result is false, the Dynamic List execution is terminated. If the same master condition is in two Dynamic Lists, both lists are skipped, if the condition was false.

In all commands explicit defaults for Data Base, node and Directories can be specified. These parameters are not included in the following descriptions:

```
DYN_DIR=default Directory of Dynamic List
COND_DIR=default Directory of condition
SPEC_DIR=default Directory of spectrum
PAR_DIR=default Directory of parameters
POLY_DIR=default Directory of polygon
BASE=default Data Base
NODE=default node
```

6.10.3 Execution

Note that for conditions, spectra and picture frames specific freeze bits may be set or cleared by commands. This disables/enables the execution of individual Dynamic List Entries without modifications of the Dynamic List itself.

The Dynamic List is executed in the following order (the `CREATE DYNAMIC ENTRY` subcommand keys are given in parenthesis):

1. Master procedures (PROCEDURE /MASTER)
   Call specified user written procedures (modules in sharable images).
2. Master pattern conditions (PATTERN /MASTER)
   Execute pattern condition test, return if false.
3. Master window conditions (WINDOW /MASTER)
   Execute window condition test, return if false.
4. Master function conditions (FUNCTION /MASTER)
   Call specified user function, return if false.
5. Master polygon conditions (POLYGON /MASTER)
   Check polygon, return if false.

6. Master composed conditions (COMPOSED /MASTER)
   Execute composed condition test, return if false.
7. Procedures (PROCEDURE)
   Call specified user written procedures (modules in sharable images).
8. Pattern conditions (PATTERN)
   Execute pattern condition test.
9. Multi Window conditions (MULTI)
   Execute multi window condition test.
10. Window conditions (WINDOW)
    Execute window condition test.
11. Function conditions (FUNCTION)
    Call specified function (module in sharable image).
12. Polygon conditions (POLYGON)
    Check polygon.
13. Composed conditions (COMPOSED)
    Execute composed condition test.
14. Spectrum accumulation (SPECTRUM)
    Accumulates 1-2 dimensional spectra of type L,R.
15. Spectrum accumulation indexed (INDEXEDSPECTRUM)
    Accumulates 1-2 dimensional spectra of type L,R.
16. Bit spectrum accumulation (BITSPECTRUM)
    Accumulates 1 dimensional bit spectra of type L,R.
17. Scatter plots (SCATTER)
    Send buffered scatter parameter data to displays.

6.10.4 Arrays

Spectra or conditions may be name arrays. In this case an index range must be specified. All referenced Data Elements must be either scalar or indexed by the same range. Ranges are specified by (lower limit : upper limit).
Examples:

```
CRE DYN ENTRY WINDOW dlist [d]e_recoil(1:5)
          PARA=[d]$event.ener
CRE DYN ENTRY SPECTRUM dlist [d]ener1(2:4)
```

```
           PARA=[d]$event.e(2:4) CONDI=[d]de_window
CRE DYN ENTRY SPECTRUM dlist [d]ede(1:4)
           PARA=([d]$event.e,$event.de)
CRE DYN ENTRY INDEXED dlist [d]ede(1:7)
           PARA=([d]$event.e,$event.de)
           INDEX=[d]a.b(1)
```

[d] is the Directory specification
The difference between windows and multiwindows is that multiwindows have only one object for all subwindows, but one result bit for each, whereas windows need one object per subwindow, but have only one result bit (set, if all subwindows are true). Multi windows may be used as filters for spectrum array accumulation. The internal dimension of the window must match the specified index range. It may also be used for 'indexed' spectrum accumulation. Then the index of the last matching subwindow is used to select the spectrum member. In the first case, the subwindows may overlapp, in the second case this makes normally no sense.

```
CRE DYN ENTRY SPECTRUM list [d]ener1(2:4)
           PARA=[d]$event.e(2:4) CONDI=[d]m_window
! three spectrum Entries are executed
CRE DYN ENTRY INDEXEDSPECTRUM list [d]ener(2:4)
           PARA=[d]$event.e(1) INDEX=[d]m_window
! One spectrum Entry is executed
```

[d] is the Directory specification
In both cases 'm_window' must have 3 subwindows.

6.10.5 Entry Types

PROCEDURE

Command to insert an entry with a user specified procedure call:

```
CRE DYN ENTRY PROCEDURE listname MODULE=image(module)
                                 PARAMETER=(argument list)
                                 CONDITION=cond
                                 /MASTER
MODULE        module specification as 'image(module)'. Module
              must be linked in sharable image
image         logical name of shar.image
PARAMETER     arg.list of DE-members
CONDITION     name of condition (optional)
/MASTER       master Entry
```

This Entry will call a module from a sharable image. The pointers to the Data Elements specified in the argument list are passed to the procedure.
Example:

```
   CRE DYN ENTRY PROCEDURE dlist
                MOD=privshar(x$loop)
                PARA=([d]$event.z4.de(5),$event.z5)
                /MASTER
```

[d] is the Directory specification
The X$LOOP declaration must be:
`ENTRY(POINTER,POINTER) RETURNS(BIN FIXED(31))`
The sharable image must be linked by the DCL command `LSHARIM`.

FUNCTION

Command to insert an entry with a user specified condition function call:

```
   CRE DYN ENTRY FUNCTION listname condition MODULE=image(module)
                                   PARAMETER=(argument list)
                                   /MASTER
   MODULE        module specification as 'image(module)'. Module
                 must be linked in sharable image. Is used and required only
                 if not specified in condition.
   image         logical name of shar.image
   PARAMETER     arg.list of DE-members
   /MASTER       master entry
```

This entry will call a module from a sharable image. The pointers to the Data Elements specified in argument list are passed to the procedure. The first argument, a BIT(1) ALIGNED, returns the condition result.
Example:

```
   CRE DYN ENTRY FUNCTION dlist [d]cond
                 MOD=privshar(x$cond)
                 PARA=([d]$event.z4.de(5),$event.z5)
```

[d] is the Directory specification
The X$COND declaration must be:
`ENTRY(BIT(1) ALIGNED,POINTER,POINTER) RETURNS(BIN FIXED(31))`
The sharable image must be linked by the DCL command `LSHARIM`.

PATTERN

Command to insert a pattern condition entry:

```
   CRE DYN ENTRY PATTERN listname cond PARAMETER=object
                                       /MASTER
   PARAMETER     DE-members
   /MASTER       Master entry
```

The entry will check a specified Data Element member versus a pattern. Note that four test modes can be specified with the pattern condition (IDENT, ANY, EXCL, INCL). The values of the Data Element members can be inverted bitwise. Up to 8 internal dimensions. Objects can be of type BIT(16), BIT(32), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY PATTERN dlist [d]main_pat
         PARA=[d]$event.pat
         /MASTER
```

[d] is the Directory specification

WINDOW

Command to insert a window condition entry:

```
CRE DYN ENTRY WINDOW listname cond PARAMETER=object
                                   /MASTER
PARAMETER     DE-members
/MASTER       Master entry
```

This entry will check a specified Data Element member versus window limits. Up to 8 internal dimensions. The objects may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY WINDOW dlist [d]e_recoil
         PARA=[d]$event.ener
```

[d] is the Directory specification

MULTIWINDOW

Command to insert a multiwindow condition entry:

```
CRE DYN ENTRY MULTIWINDOW listname cond PARAMETER=object

PARAMETER     DE-member
```

This entry will check a specified Data Element member versus all window limits. For each check a result bit is set, which may be used to increment a spectrum array member. In addition, the number of the last matching window may be used as the index of a spectrum array member (see INDEXEDSPECTRUM). The object may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY MULTI dlist [d]e_recoil
         PARA=[d]$event.ener
```

[d] is the Directory specification

POLYGON

Command to insert a polygon condition entry:

```
CRE DYN ENTRY POLYGON listname cond PARAMETER=(x,y)
                                     POLYGON=name
                                     /MASTER
PARAMETER     DE-members for X and Y. Used and required only
              if not specified in condition.
POLYGON       Name of polygon. Used and required only
              if not specified in condition.
/MASTER       Master entry
```

It is checked whether the point X,Y is inside (true) or outside (false) the polygon. Objects may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).

Example:

```
CRE DYN ENTRY POLY dlist [d]poly_1
         PARA=([d]$event.de,[d]$event.ener)
         POLYG=poly
```

[d] is the Directory specification.

COMPOSED

Command to insert a composed condition entry:

```
CRE DYN ENTRY COMPOSED listname cond /MASTER

/MASTER       Master entry
```

A boolean expression of conditions is executed. The expression is specified in the corresponding condition Data Element.
Example:

```
CRE DYN ENTRY COMPOSED dlist [d]all_ok /MASTER
```

[d] is the Directory specification

SPECTRUM

Command to insert a spectrum entry:

```
CRE DYN ENTRY SPECTRUM listname spectrum PARAMETER=object
                                         CONDITION=cond
                                         INCREMENT=incr
PARAMETER     DE-members for coordinates
CONDITION     condition for spectrum (optional)
INCREMENT     DE-member for increment (optional) (BIN FLOAT(24))
```

Supports spectra of Type BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24) with up to 2 dimensions. Coordinates can be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24).
Examples:

```
CRE DYN EN SPECTRUM dlist [d]ener1
           PARA=[d]$event.e(1) CONDI=[d]de_window
CRE DYN EN SPECTRUM dlist [d]ede
           PARA=([d]$event.e,$event.de)
```

[d] is the Directory specification

INDEXEDSPECTRUM

Command to insert an indexed spectrum entry:

```
CRE DYN ENTRY INDEXEDSPECTRUM listname spectrum(l:u) PARAMETER=object
                                                     INDEX=index
                                                     INCREMENT=incr
                                                     CONDITION=cond
PARAMETER     DE-members for coordinates
INDEX         DE-member (BIN FIXED(31)) or multiwindow to specify
              spectrum member
CONDITION     condition for spectrum (optional)
INCREMENT     DE-member for increment (optional) (BIN FLOAT(24))
```

Supports spectra of Type BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24) with up to 2 dimensions. Coordinates can be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24). Specified spectrum must be an array. Specification of index is used to select the spectrum member to be incremented. This could be either a parameter Data Element or a multiwindow.
Examples:

```
CRE DYN EN INDEXED dlist [d]ener(1:10)
           PARA=[d]$event.e(1) INDEX=[d]m_window
CRE DYN EN INDEXED dlist [d]ede(1:5)
           PARA=([d]$event.e,$event.de) INDEX=[d]a.b
```

[d] is the Directory specification

BITSPECTRUM

Command to insert a bitspectrum entry:

```
CRE DYN ENTRY BITSPECTRUM listname spectrum PARAMETER=object
                                            CONDITION=cond
PARAMETER     DE-members for coordinates
CONDITION     condition for spectrum (optional)
```

Supports one dimensional spectra, Type BIN FIXED(31). Coordinates must be BIT(32), BIT(16), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY BIT dlist [d]patt
          PARA=[d]$event.pat(1)
```

[d] is the Directory specification

SCATTER

This entry is inserted by the `DISPLAY PICTURE` command, if scatter frames are in the picture, or by the `DISPLAY SCATTER` command. The name of the list can be specified optionally with these commands. The default is $SCATTER. Note that this list must be attached to be active. It should be attached as the last list. Scatter Entries are deleted only by the creating display process. This may lead to 'dead' scatter Entries, i.e. if the environment name is no longer used. Attaching the list in this case a message is displayed that a link could not be opened. Then one should delete all scatter Entries of all types by the command:

```
DELETE DYNAMIC ENTRY SCATTER list * * /UPDATE
```

No scatter plot should be active during that command.

6.11 GOOSY Control

An example of an ONLINE session is given in section 6.14 on page 132).

6.11.1 Commands

We give a short summary of most often used commands to control the data taking and data analysis. First some data taking control commands:

```
LOAD MBD or J11 or STARBURST           ! Load frontends
INIT ACQUIS /J11 or /MBD or /FILE   ! Select input stream
START or STOP ACQUISITION    !
START or STOP OUTPUT FILE    ! Writing to tape/disk from data acquisition
SHOW ACQUISITION             !
SHOW STARBURST               !
TYPE BUFFER                  ! Stop output by <CTRL>O
TYPE EVENT n                 ! Type n events on the screen
```

Here some analysis control commands:

```
START,STOP INPUT FILE     ! Read offline from file (disk or tape)
START,STOP INPUT MAILBOX ! Read online from Transport Manager (CAMAC)
                          ! The buffer size must be specified!
START,STOP INPUT NET      ! Read online from a Transport Manager on a remote VAX
START,STOP ANALYSIS OUTPUT  ! Write to disk or tape
START,STOP SCATTER        !
START,STOP DYNAMIC LIST   !
SET SCATTER BUFFER        ! Set scatter buffer size in display points
SHOW ANALYSIS             !
SHOW DYNAMIC ATTACHED     ! Actual executing lists ! analysis
```

Now some more DBM commands:

```
SHOW DYNAMIC LIST         ! As defined in Data Base
SHOW TABLE                ! Condition and spectra counters
SHOW GOOSY STATUS
SHOW CONDITION
SHOW SPECTRUM
SHOW ELEMENT
SHOW MEMBER
[UN]FREEZE SPECTRUM       ! Inhibit the accumulation into a spectrum
[UN]FREEZE CONDITION      ! Inhibit the execution (set) of a condition
```

6.11.2 Batch and DCL Procedures

All GOOSY commands may be executed in DCL procedures mixed with DCL commands. Because any DCL procedure may run as a batch job, there are no special precautions to run GOOSY in batch. Note, that the environment name must be unique on each VAX node within a user group. There is one DCL implemented command needed to synchronize an executing analysis with the GOOSY prompter, because the analysis executes in parallel to the main session. This command is `MGOOWAIT process`.

```
$ GOOSY ATTACH DYNAMIC LIST l1   ! activate Dynamic List
$ GOOSY START INP FILE xxx       ! start analysis
$ MGOOWAIT GN_env____$anl        ! Wait
$ GOOSY SHOW SPECTRUM /TABLE     ! Show spectrum contents
```

`MGOOWAIT` hibernates until the analysis stops by end of file.

6.11.3 File Output

Writing list mode data from Transport Manager to a file must be explicitly started and stopped by commands. After opening an output file, file writing can be started and stopped without closing the file. The acquisition can be started and stopped without affecting the file status. The file can be on disk or on tape. A tape to be used must be mounted. The file is filled regardless of analysis programs.

START-STOP OUTPUT FILE

To start file output to a new file and close a file one types

```
GOOSY> START OUTPUT FILE file size number /OPEN/AUTOMATIC
GOOSY> STOP OUTPUT FILE /CLOSE
```

Then a new file is opened. 'Size' specifies the intended size of the file. When the file is filled it is automatically closed and the acquisition is stopped. All data from the frontends are transferred to the file. Then the file is closed. 'Number' is optionally used together with the `/AUTO` switch. It means that 'number' files of size 'size' are automatically opened, filled, and closed. A running number is added to the file name in this case. The `/OPEN` and `/CLOSE` qualifiers are default. To stop file writing and to start writing the same file type

```
GOOSY> STOP OUTPUT FILE /NOCLOSE
GOOSY> START OUTPUT FILE /NOOPEN
```

The STOP OUTPUT FILE commands does not stop the acquisition.

GOOSY File Header

A GOOSY file header is written to each file after it had been opened. The header information can be prompted or read from a text file (see command description of `START OUTPUT FILE`).

File Names

If one wants to send the output files to the IBM, the filenames must follow some conventions:

1. Maximal length 25 char (including type)
2. Maximal 8 char or 7 digits between two underscores (No $).
3. File type must be .LMD

Tape Handling

Writing to tape requires some additional operations. If the tape is new, it must first be initialized and then mounted. The initialization and the mount should be done within the TMR.

```
   GOOSY> MOUNT TAPE tape-device tapename /INIT
```

With these commands the tape density and the size of the tape records can be specified. The defaults should be adequate. The name of the tape is used for the (optional) initialization. After that the tape file will be opened and started like a disk file. You may specify the device together with the file name or as a separate parameter. In the last case the device will be defaulted for following commands. **Use STOP ACQUISITION before STOP OUTPUT FILE and START OUTPUT FILE before START ACQUISITION to make sure that all data sent from CAMAC are written to the file!** If a file size limit is specified, the acquisition is stopped automatically early enough to write all buffers to the file.

To dismount the tape issue the GOOSY command:

```
  GOOSY> STOP OUTPUT FILE /CLOSE
  GOOSY> DISMOUNT TAPE device:
```

End of Tape

When the tape runs out of space, a STOP ACQUISITION is executed and the file is closed. All data from the frontends are transferred to the file! However, when the tape was not empty at the beginning, the TMR cannot know the space available on the tape. In this case it may happen, that the tape end is reached. Then OpenVMS rewinds the tape and requires a continuation tape. Mount the next tape on the device and look to the VAX operator console for the number of your tape request. Then type on your terminal in DCL:

```
     $ NEXTTAPE number
```

6.11.4 Analysis Control

Independent of a Transport Manager the analysis may be started an stopped. The analysis gets data via a mailbox, a DECnet channel or from a file. It also may write output events to a file and/or DECnet.

```
START INPUT MAIL
START INPUT NET node envir component
START INPUT FILE filespec.
```

A more detailed description can be found in the 'GOOSY Data Acquisition and Analysis' manual.

6.11.5 CAMAC Esone Calls

The Transport Manager provides some commands to execute CAMAC functions and to test specific CAMAC modules. The commands are:

```
CAMAC CLEAR
CAMAC CNAF
CAMAC DEMAND
CAMAC INHIBIT
CAMAC INITIALIZE
CAMAC SCAN
TEST module
```

If using a single crate system, a logical name `CAMAC_BRANCH_0` must be defined:

```
$ DEFINE CAMAC_BRANCH_0 "J11x::GESONE()"
```

where 'x' is the node specific part of the node name of the J11. A more detailed description can be found in the 'GOOSY Hardware' manual.

6.12 GOOSY Display

6.12.1 Display Devices

Before displaying spectra or pictures it is necessary to connect a graphical output device to the display process. This could be done with the `ALLOCATE DEVICE` GOOSY display command (**not** the DCL command ALLOCATE):

```
ALLOCATE DEVICE name type /[NO]MAIN
```

Each device has an identifying name which is the physical device address. If it is your current command terminal, the '`name`' is **TT**. The '`type`' is a GOOSY alias name specifying the terminal type. The following terminal types are supported:

MG600 Monterey MG600 and MG620 terminals (default)

PECAD PECAD terminals

TEK4014 TEKTRONIX 4014 or compatibles

TEK4107 TEKTRONIX 4107 or compatibles

TEK4109 TEKTRONIX 4109 or compatibles

TEK4111 TEKTRONIX 4111 or compatibles

TEK4115 TEKTRONIX 4115 or compatibles

Motif VAX/AXP workstations or X-terminals.

MV,VS,GPX Micro VAX VAXstation display (black and white, GPX) a new window will be created with the window-name 'name'.

VT240 DEC VT240, VT241, VT330, VT340 or compatible Regis terminals.

LN03 LN03-PLUS laser-printer. A temporary plotfile `name`.LN3 is generated. The file can be printed later under DCL, e.g. by `$ PCTEK name.LN3`.

POST Postscript. A temporary plotfile `name`.PS is generated. The file can be printed later under DCL.

HP7550A4 Pen-plotter HP7550 with DIN A4 sheets. A temporary plotfile `name`.HP4 is generated. The plotfile can be copied later to a VAX-HP-plotter.

HP7550A3 Pen-plotter HP7550 with DIN A3 sheets. A temporary plotfile `name`.HP3 is generated. The plotfile can be copied later to a VAX-HP-plotter.

METAOUT Generates a device independent plotfile `name`.MET. The metafile can be plotted later by the DCL command `$ PLOTMET`, see also Help and GOOSY command `PLOT METAFILE`.

In GOOSY up to 10 different graphical devices could be allocated simultanously. Therefore one GOOSY main device must be specified from which all graphical **inputs** are performed. This is done with the `/MAIN` switch. As default the first allocated device is used as the GOOSY main device. All devices allocated simultanously will get the identical graphics output. You cannot use them independently. To do this, you must start a second display program on another command terminal.

- **Examples:**

```
ALLOCATE DEVICE txa2           ! terminal at physical address TXA2: is
                               ! allocated as a MONTEREY MG600
                               ! or a MG620 (default).
ALLOCATE DEVICE plotter ln03   ! A plotfile PLOTTER.LN3 is generated
                               ! which could be printed on any
                               ! LN03-PLUS laser printer.
ALLOCATE DEVICE scatter metaout ! A device independent plotfile
                               ! SCATTER.MET is created.
```

6.12.2 Displaying Pictures, Spectra and Scatterplots

As mentioned above Pictures are groups of spectra and scatter plots that are displayed together. Normally, pictures are created during the Data Base creation phase in a DCL procedure. The command syntax for displaying pictures or spectra is very similar

```
DISPLAY PICTURE  name     ! Picture "name" is displayed .
DISPLAY PICTURE  name/LOG ! All frames are displayed in log-mode
DISPLAY PICTURE           ! The last displayed picture is displayed again.
DISPLAY SPECTRUM name     ! Spectrum "name" is displayed.
                          ! "name" could be one or two dimensional.
DISPLAY SPECTRUM name/LOG ! The spectrum is displayed in log-mode.
DISPLAY SPECTRUM          ! The last displayed spectrum is displayed again.
```

A single scatterplot could be displayed with

```
DISPLAY SCATTER x y (xlow,xhigh,ylow,yhigh) cond
```

where `x` and `y` are the parameters displayed on the x and y axis within the limits `xlow` to `xhigh` and `ylow` to `yhigh`. An optional condition `cond` may be specified. The default Directory for the parameters is `data` and may be ommitted in the specification. Similar the default Directory for a condition is `$CONDITION`. The condition must be executed either in the analysis program or in a dynamic list. If the scatter plot should be executed in a dynamic list different than `$SCATTER`, this list must be specified:

```
DISPLAY SCATTER x y (xlow,xhigh,ylow,yhigh) DYN_SCAT=list
```

NOTE that the dynamic list must be attached. **Leave the display menu after activating any scatter plots!**.

For a more convenient way to execute these display commands, there are special keys defined for them:

$\boxed{\texttt{KP\_1}}$ for DISPLAY SPECTRUM

$\boxed{\texttt{KP\_2}}$ for DISPLAY SCATTER

$\boxed{\texttt{KP\_3}}$ for DISPLAY PICTURE

Press these keys followed by parameters as shown above.

6.12.3 Calibrating Spectra

Spectra can be displayed applying any kind of calibration to them. In GOOSY it is possible to draw a calibrated axis, by which the distance between the tics need not be equal. Or the spectrum itself can be calibrated, then the size of the bins may vary.

```
DISPLAY SPECTRUM name /CALAX       ! Draw a calibrated axis
DISPLAY PICTURE name /CALAX        ! for each calibrated spectrum
DISPLAY SPECTRUM name /CALSP       ! Calibrate the spectrum
DISPLAY PICTURE name /CALSP        ! in each frame
DISPLAY SPECTRUM name/CALAX/NOCHAN ! only the calibrated axis is shown
```

The calibrations are kept in tables, i.e. specific Data Elements in the Data Base. Three types of calibrations are supported:

LINEAR calibration with a linear polynom, the polynom parameters are kept in the Data Element.

FIXED calibration table with a fixed step width between the uncalibrated values.

FLOAT calibration table with an arbitrary stepwidth between the uncalibrated entries.

The calibration tables are created and set with the following commands:

```
CREATE CALIBRATION LINEAR name
CREATE CALIBRATION FIXED name 1024 ! FIXED-type calibration with
                                   ! 1024 entries
CREATE CALIBRATION FLOAT name 1024 ! FLOAT-type calibration with
```

```
                                       ! 1024 entries
SET CALIBRATION LINEAR
SET CALIBRATION FIXED name unit
SET CALIBRATION FLOAT name unit
```

A calibration can be connected to any number of spectra with the command

```
CALIBRATE SPECTRUM spectrum calibration
```

Before a spectrum can be deleted, the links to the calibration must be canceled by the command

```
DECALIBRATE SPECTRUM spectrum
```

6.12.4 Temporary Display Modes

All switches specified together with display commands are **temporary display modes**. They are available only until the next DISPLAY command is given.

```
DISPLAY SPECTRUM name/LOG/MARKER
                             ! LOG mode on scaling axis
                             ! spectrum is displayed with markers.
DISPLAY SPECTRUM name/TEMP ! The spectrum is displayed with the
                             ! display modes specified in the last
                             ! DISPLAY PICTURE or DISPLAY SPECTRUM
                             ! command that means in LOG and MARKER
                             ! mode
DISPLAY PICTURE name/TEMP/HISTO
                             ! LOG mode on scaling axis
                             ! But spectra are displayed as histogramms.
```

6.12.5 Global Display Modes

Sometimes it is useful to display all spectra or pictures with the same display modes. For spectrum frames and pictures these **global modes** can be defined separatly

```
DEFINE DISPLAY PICTURE /LOG/XLOG/SMOOTH/CLUSTER
                            ! The specified switches are stored
                            ! as global parameters for pictures
DEFINE DISPLAY SPECTRUM /SQRT/XSQRT/VECTOR
                            ! The specified switches are stored as
                            ! as global parameters for spectra
```

These global definitions can be generally (de)activated by

```
DEFINE DISPLAY PICTURE /[NO]ACTIVE
DEFINE DISPLAY SPECTRUM /[NO]ACTIVE
```

The global settings can be activated or suspended for just one command by

```
DISPLAY PICTURE name /[NO]GLOBAL
DISPLAY SPECTRUM name /[NO]GLOBAL
```

If global modes are active, single mode parameters can be overwritten for just one command:

```
DISPLAY SPECTRUM name/LIN ! The global display modes for spectra
                          ! are activated, but the scaling axis is
                          ! displayed in LIN-mode.
```

To save CPU time there are two modes of the display. They are selected by

```
SET DISPLAY MODE /STANDARD
SET DISPLAY MODE /FAST
```

The fast version saves CPU time. If the CPU is busy, i.e. by an analysis, this version will result in a faster display. This version, however, **does not** provide the `PLOT` and `SAVE` commands. The version to be used can be changed at any time using the above commands.

6.12.6 Setting Window Conditions

Condition windows could be set with the `REPLACE CONDITION WINDOW` and `SET CONDITION WINDOW` commands. Only the `REPLACE` command provides cursor input.

```
SET CONDITION WINDOW name (low,up) dim
                      ! Condition limits "low" and "up"
                      ! of condition "name" in dimension
                      ! "dim" are set.
REPLACE CONDITION WINDOW name (low,up) dim
                      ! Condition limits "low" and "up"
                      ! of condition "name" in dimension
                      ! "dim" are set.
REPLACE CONDITION WINDOW name
                      ! Cursor appears to set limits in
                      ! condition "name" in all dimensions.
```

6.12.7 Useful DISPLAY Commands

In the following the most commonly used display commands will be listed and explained briefly.

If the displayed range of a spectrum or the range in one frame in a picture is too large it could be changed by

```
EXPAND low,up frame/CAL ! Frame "frame" is expanded within the
                        ! limits low and up. The limits are given
                        ! in calibrated units.
EXPAND                  ! The cursor appears to get limits in
                        ! one frame.
```

Spectra could be integrated by

```
INTEGRATE low,up frame ! Spectrum in frame "frame" is integrated
                 /CHAN ! within "low" and "up". The limits are given
                       ! in channels.
INTEGRATE              ! The cursor appears to get limits in
                       ! every frame on the screen.
INTEGRATE /LOOP        ! Enter cursor loop to get several integration
                       ! ranges in every frame on screen.
```

Unfortunately, it takes always some time to create the Dynamic List Entries with the `DISPLAY PICTURE` command for scatterplots defined in a picture Data Element. If the same picture should be displayed again, it is faster to refresh the screen with

```
REFRESH frame          ! A single frame is deleted on screen
                       ! and will be redrawn with the old spectrum
                       ! contents. The old scatterplot information
                       ! will be lost.
REFRESH *              ! The whole screen is cleared and all frames are
                       ! redrawn. The scatterplot informations are lost.
```

A two dimensional spectrum is projected to one dimension by

```
PROJECT SPECTRUM s2 s1 (100,200) ! Limits may be specified by
                                 ! cursor or condition
```

The organisation of the picture frames on the screen and the size of the single frames could be changed by two commands

```
DEFINE PICTURE SETUP
DEFINE FRAME SETUP
```

for details look at the command description.

6.12.8 Plotting Pictures, Spectra, and Scatterplots

The following commands work only in the standard display mode, but not in the fast mode. Use the `SET DISPLAY MODE/STANDARD` command if you are in `/FAST` mode and display the picture to be plotted. The final display on the screen could be sent to a plotter with

```
PLOT PICTURE queue type  ! The displayed picture is plotted
                         ! on the plotter "queue" (e.g. SYS$LN03_C)
                         ! of type "type" (e.g. LN03).
```

This command plots only information kept in the local display memory. Scatterplot points are **not** kept in the memory. That means that scatterframes on the screen are empty on the plotter output.

If you like to plot scatterframes you first have to allocate a plotter (see section 6.12.1 on page 124) or a device independent plotfile, called metafile. All graphical outputs are then sent to these files. To get a scatter picture on laser printer, use the following command sequence:

```
ALLOCATE DEVICE filename LN03  ! Open printer file
DISPLAY SCATTER ...            ! Display scatterplot or picture
DEALLOCATE DEVICE filename     ! Close printer file
$ PATEX filename               ! Send file to printer LN03_A
```

Plotfiles or metafiles are generated by the `ALLOCATE DEVICE` command. A device dependent plotfile or a device independent metafile can be plotted with

```
PLOT PLOTFILE file queue
PLOT METAFILE file queue type
```

where the queue depends on the output device, e.g. 'SYS$LN03_C' or for `PLOT METAFILE` also 'IBM::'. The type is the device name, e.g. LN03 or for IBM the RP01 or RP02:

```
PLOT METAFILE xxx.MET SYS$LN03_C LN03  !xxx will be plotted on laser printer C
PLOT METAFILE xxx.MET IBM:: RP02       !xxx will be plotted on IBM rasterplotter
```

Metafiles can also be plotted from DCL by command `PLOTMET`.

6.13 Error Recovery

GOOSY modules use the VAX-OpenVMS error handling utilities. GOOSY error messages are formatted and output as OpenVMS error messages, i.e.:

```
%facility-severity-code, message text
%GOODM-E-ALREX, M$CRESP: spectrum already exist in data base.
```

The 'severity' is a letter indicating the error level of the message. It can be **S**uccess, **I**nformation, **W**arning, **E**rror, or **F**atal. In many cases one may get more information about a GOOSY error by the DCL command:

```
$ HELP @REC facility code
$ HELP @REC GOODM ALREX
```

'facility' and 'code' can be obtained from the message output. There is also a manual containing all recovery information. GOOSY outputs an additional headline for each message containing the module name, user name, node name, date etc.. To make the reading of the error messages more convenient, one may suppress the output of this line and/or the output of the message prefix (%facility-severity-code). This is done by:

```
 $ SETMESSAGE GOOSY /NOPREF/NOHEAD
 $ SETMESSAGE GOOSY
```

The second line enables full message output again.

6.14 Session Examples

In the following we show a few examples of ONLINE or OFFLINE GOOSY sessions. The examples can be taken literally. Some parameters, however, have to be replaced by actual values because they depend on the CAMAC setup, the DECnet node where the J11, runs and the display device. The environment name must be specified, too. All these names to be adapted are marked by <>. The example files are on Directory GOO$EXAMPLES: and can be copied from there.

6.14.1 Preparations for the Examples

The following three steps are common to all examples. We assume that the LOGIN.COM procedure is already prepared for GOOSY (see section 6.2.4 on page 73).

Creating Data Elements

The standard event Data Element of J11 generated data is provided by GOOSY (library GOOTYP) and has the structure:

```
    /* ================= GSI Event header ======================= */
    DCL P_SA$event       POINTER;
    DCL 1 SA$event         BASED(P_SA$event),
2 IA$event_dlen    BIN FIXED(15),
2 IA$event_tlen    BIN FIXED(15),
2 IA$event_type    BIN FIXED(15),
2 IA$event_subtype BIN FIXED(15),
2 IA$event(512)    BIN FIXED(15);
    /*-------------------------------------------------------------*/
```

The ADC data are stored by the GOOSY unpack routine in IA$EVENT. The first ADC is in IA$EVENT(1:8), the second in IA$EVENT(9:16). A third SILENA would be in IA$EVENT(17:24) and so on.

We want to use a Data Element for calibrated data (See section 6.5 on page 86). The declaration is:

```
/* file s_out.txt:        */
DCL P_out POINTER;
DCL 1 S_out BASED(P_out),
    2 R_energy BIN FLOAT(24),
    2 R_time   BIN FLOAT(24);
```

Note that the name of the file is the name of the structure. We put it into our private text library TPRIV (defined in LOGIN.COM). From there it can be included in PL/1 procedures. It also can be used to create a GOOSY Data Element in the Data Base.

```
   $ LIB/REPL TPRIV S_out.txt    ! store module in a text library
```

Creating the Data Base

Now we have to create a Data Base. We write a command procedure to do that (See sections 6.4 on page 82, 6.5 on page 86, 6.6 on page 91, and 23.9 on page 322).

```
$! file credb_j11.com
$!
$! Data Element: event (Type SA$event)
$!                newevent (Type S_out)
$! Spectra:   single(1:16)
$!            energy
$!            time
$!            window
$!            pattern
$! Condition: peak
$! Dyn.list:  accu
$!
$ ON ERROR THEN CONTINUE
$ CREDB db db_j11.sec 2000 /NEW
$! The Data Base name 'db' is specifed once and
$! defaulted in the following commands.
$ MDBM
CREATE TYPE      "@GOOTYP(SA$event)"
CREATE TYPE      "@TPRIV(S_out)"
CREATE ELEMENT event    TYPE=SA$event DIR=data POOL=data
CREATE ELEMENT newevent TYPE=S_out    DIR=data POOL=data
CREATE SPECTRUM single(16) L (0,2047) /DIGITAL
CREATE SPECTRUM window     L (0,2047) /DIGITAL
CREATE SPECTRUM pattern    L (1,32)   /DIGITAL
CREATE SPECTRUM energy     R (0,2047) /ANALOG
CREATE SPECTRUM time       R (0,2047) /ANALOG
CREATE CONDITION WINDOW peak (1150,1200)
CREATE DYNAMIC LIST accu ENTRIES=20
CREATE DYNAMIC ENTRY WINDOW accu peak PARA=event.IA$event(4)
CREATE DYNAMIC ENTRY SPECTRUM accu single(1:16) PARA=event.IA$event(1:16)
CREATE DYNAMIC ENTRY SPECTRUM accu window PARA=event.IA$event(7) CONDI=peak
CREATE PICTURE raw 8 /NOPROMPT
MODIFY FRAME SPEC raw 1 single(1)
MODIFY FRAME SPEC raw 2 single(2)
MODIFY FRAME SPEC raw 3 single(3)
MODIFY FRAME SPEC raw 4 single(4)
MODIFY FRAME SPEC raw 5 single(5)
MODIFY FRAME SPEC raw 6 single(6)
MODIFY FRAME SPEC raw 7 single(7)
MODIFY FRAME SPEC raw 8 single(8)
DISMOUNT BASE db
$ EXIT
```

Writing the Analysis Routine

Now we write an analysis routine (See section 6.9 on page 99). This routine does the same as the Dynamic List. In addition it fills spectra 'energy' and 'time'. If we were satisfied with the Dynamic List we would not need it.

```
/* file X$ANAL.PPL                       */
/*****************************************/
X$ANAL:@PROCEDURE RETURNS(BIN FIXED(31));
/*****************************************/
@INCLUDE $MACRO(dcl_proc);
@INCLUDE $MACRO(S$mess);
@INCLUDE $MACRO($MACRO);
@INCLUDE $MACRO(SA$event);
@INCLUDE $MACRO(S_out);

DCL P_event    POINTER STATIC;
DCL P_newevent POINTER STATIC;
DCL B_peak     BIT(1) ALIGNED;
DCL L_index    BIN FIXED(31);
DCL L_incr     BIN FIXED(31) STATIC INIT(1);

@ON_ANY_W(U_CLEANUP);           /* Error routine */

P_SA$event = P_event;           /* Get pointer to input event */
P_out      = P_newevent;        /* Get pointer to output event */

STS$value=1;
/* The data is in IA$event(1:16) */
$COND(WC,DB,$CONDITION,peak,B_peak,1,IA$event(4));
DO L_index = 1 TO 16;
    $ACCU1(L,DB,$SPECTRUM,single,L_index,L_incr,1,IA$event(L_index));
    END;
IF B_peak THEN DO;
    S_out.R_energy=(FLOAT(IA$event(4),24) + FLOAT(IA$event(7),24))/4.;
    S_out.R_time=(FLOAT(IA$event(4),24) + FLOAT(IA$event(7),24))/2.;
    $ACCU(L,DB,$SPECTRUM,window,L_incr,1,IA$event(7));
    $ACCU(R,DB,$SPECTRUM,energy,L_incr,1,S_out.R_energy);
    $ACCU(R,DB,$SPECTRUM,time,L_incr,1,S_out.R_time);
    END;

@RET(STS$value);
/****************************************************/
/* Initialization to locate all used Data Elements  */
/****************************************************/
$XANAL:ENTRY RETURNS(BIN FIXED(31));
@INCLUDE $MACRO($SECDEF);

$LOC(DE,db,data,event,W);
IF ^STS$success THEN @RET(STS$value);
P_event=P$_DB_DATA_EVENT;

$LOC(DE,db,data,newevent,W);
IF ^STS$success THEN @RET(STS$value);
P_newevent=P$_DB_DATA_NEWEVENT;

$LOC1(SPEC,DB,$SPECTRUM,single,1,16,W,L);
IF ^STS$success THEN @RET(STS$value);
$LOC(SPEC,DB,$SPECTRUM,energy,W,R);
IF ^STS$success THEN @RET(STS$value);
$LOC(SPEC,DB,$SPECTRUM,time,W,R);
IF ^STS$success THEN @RET(STS$value);
$LOC(SPEC,DB,$SPECTRUM,window,W,L);
IF ^STS$success THEN @RET(STS$value);
$LOC(COND,DB,$CONDITION,peak,W,WC);
IF ^STS$success THEN @RET(STS$value);

@RET(STS$value);
/*************************************************/
/* This routine is called in case of an error    */
/*************************************************/
U_CLEANUP:PROCEDURE;
END U_CLEANUP;
END X$ANAL;
```

We compile the analysis routine and link our analysis program by the DCL commands:

```
$ CHANAL X$ANAL.PPL  !check macro calls
$ COMP X$ANAL        !compile
$ LANL X$ANAL        !make MGOOANL.EXE
```

You may now continue one of the following examples.

6.14.2 OFFLINE Analysis

Instead of getting the data from the TMR, the Analysis Manager may read the data from a file. The Data Base and analysis routine X$ANAL need not to be modified. This example can be executed literally except the environment name which should be different. The startup looks like

Procedures Starting GOOSY

We write a small command procedure to startup GOOSY.

```
$! file anl_startup.com
$ IF p1 .EQS. "" THEN INQUIRE p1 "Enter environment name "
$ MDBM MOUNT BASE db db_j11.sec
$ CRENV 'p1' /$DSP/$DBM/$ANL  ! No TMR
$ GOOSY CLEAR SPECTRUM *
$ EXIT
```

Similar command procedure to shutdown GOOSY:

```
$! file anl_shutdown.com
$ GOOSY
  STOP INPUT FILE/CLOSE
$ DLENV      ! delete environment
$ MDBM DISMOUNT BASE db
$ EXIT
```

Procedures Starting the Analysis

We write a small command procedure to start the analysis:

```
$! file anl_start.com
$ GOOSY
ATT DYN LIST accu           ! activate the Dynamic List 'accu'
SET ANAL/NOANAL             ! disable X$ANAL in the event loop
START INPUT FILE DAY$ROOT:[GOOFY]J11.LMD/OPEN ! start reading file
$ EXIT
```

Similar command procedure to stop the analysis:

```
$! file anl_stop.com
$ GOOSY
STOP INPUT FILE
DET DYN LIST accu
$ EXIT
```

Command Sequences

Now a typical session could proceed as follows:

```
$ @credb_j11         ! create Data Base
$ @anl_startup <env> ! create GOOSY environment
$ @anl_start         ! start GOOSY (dyn.list only)
$ GOOSY              ! enter GOOSY prompter
GOOSY> SHOW ANAL
GOOSY> ALL DEV <dev> PECAD/MAIN   ! <dev> is the display terminal name
GOOSY> DISP PI raw
GOOSY> DISP SP energy
GOOSY> DISP SP time
GOOSY> DISP SP single(1)
GOOSY> REPLACE COND WIND peak ! set condition by cursor
GOOSY> STOP INPUT FILE/CLOSE ! If we don't want to continue
GOOSY> CLEAR SPECTRUM *      ! clear all spectra
GOOSY> DETACH DYN LIST accu  ! disable dyn.list
GOOSY> SET ANAL/ANAL         ! enable X$ANAL
GOOSY> START INP FIL DAY$ROOT:[GOOFY]J11.LMD/OPEN ! open again if we closed it
GOOSY> <CTRL>Z               ! leave GOOSY prompter
$ @anl_stop
$ @anl_shutdown
```

6.14.3 OFFLINE Analysis with Transport Manager

Instead of getting the data from the J11, the Transport Manager may read the data from a file. The Data Base and the analysis routine X$ANAL need not to be modified. To analyze data one does not need necessarily a TMR because the Analysis Manager can read the data from a file directly. The present example can be useful to get familiar with the TMR. It can be executed literally except the environment name which should be different.

Procedures Starting GOOSY

We write a small command procedure to startup GOOSY.

```
$! file off_tmr_startup.com
$ IF p1 .EQS. "" THEN INQUIRE p1 "Enter environment name "
$ MDBM MOUNT BASE db db_j11.sec
$ CRENV 'p1' /$TMR/$DSP/$DBM/$ANL
$ GOOSY INI ACQUIS /FILE       ! read data from file instead of from J11
$ GOOSY CLEAR SPECTRUM *       ! clear all spectra
$ EXIT
```

Similar command procedure to shutdown GOOSY:

```
$! file off_tmr_shutdown.com
$ DLENV      ! delete environment
$ MDBM DISMOUNT BASE db
$ EXIT
```

Procedures Starting the Analysis

We write a small command procedure to start the acquisition:

```
$! file off_tmr_start.com
$ GOOSY
START INPUT MAIL             ! Start reading data from mailbox
ATT DYN LIST accu            ! activate the Dynamic List 'accu'
SET ANAL/NOANAL              ! disable X$ANAL in the event loop
SET ACQUIS/SYNC/MAIL         ! Synchronize input with analysis
OPEN FILE DAY$ROOT:[GOOFY]J11.LMD ! open file for TMR input
START ACQUIS                 ! Start reading file
$ EXIT
```

Similar command procedure to stop the acquisition:

```
$! file off_tmr_stop.com
$ GOOSY
STOP ACQUIS
CLOSE FILE
STOP INPUT MAIL
DET DYN LIST accu
$ EXIT
```

Command Sequences

Now a typical session could proceed as follows:

```
$ @credb_j11                ! create Data Base
$ @off_tmr_startup <env>    ! create GOOSY environment
$ @off_tmr_start            ! start GOOSY (dyn.list only)
$ GOOSY                     ! enter GOOSY prompter
GOOSY> SHOW ACQUIS
GOOSY> SHOW ANAL
GOOSY> TYPE EVENT 5/SAMPLE
GOOSY> ALL DEV <dev> PECAD/MAIN   ! <dev> is the display terminal name
GOOSY> DISP PI raw
GOOSY> DISP SP energy
GOOSY> DISP SP time
GOOSY> DISP SP single(1)
GOOSY> REPLACE COND WIND peak ! set condition by cursor
GOOSY> STOP ACQUIS
GOOSY> CLOSE FILE           ! if we don't want to continue
GOOSY> CLEAR SPECTRUM *     ! clear all spectra
GOOSY> DETACH DYN LIST accu ! disable dyn.list
GOOSY> SET ANAL/ANAL        ! enable X$ANAL
GOOSY> OPEN FILE DAY$ROOT:[GOOFY]J11.LMD ! open again if we closed it
GOOSY> START ACQUIS
GOOSY> <CTRL>Z              ! leave GOOSY prompter
$ @off_tmr_stop
$ @off_tmr_shutdown
```

6.14.4 ONLINE with a Single CAMAC Crate

We show an example of a trivial experiment using a single CAMAC crate with two SILENA ADC's. The CAMAC is controlled by a J11 auxiliary crate controller. The hardware setup for such a system is described in detail in the GOOSY Hardware Manual.

CAMAC Setup File

First we have to write a CAMAC setup file to describe the modules in the crate. This file must be adapted to the actual CAMAC setup. The example describes two SILENA ADC's in station N8 and N9, respectively.

```
! file j11c.cam, the crate number is not yet used
C=0, N=8, A=0, TY=STANDARD
C=0, N=8, A=1, TY=STANDARD
C=0, N=8, A=2, TY=STANDARD
C=0, N=8, A=3, TY=STANDARD
C=0, N=8, A=4, TY=STANDARD
C=0, N=8, A=5, TY=STANDARD
C=0, N=8, A=6, TY=STANDARD
C=0, N=8, A=7, TY=STANDARD
C=0, N=9, A=0, TY=STANDARD
C=0, N=9, A=1, TY=STANDARD
C=0, N=9, A=2, TY=STANDARD
C=0, N=9, A=3, TY=STANDARD
C=0, N=9, A=4, TY=STANDARD
C=0, N=9, A=5, TY=STANDARD
C=0, N=9, A=6, TY=STANDARD
```

```
C=0, N=9, A=7, TY=STANDARD
```

Procedures Starting GOOSY

We write a small command procedure to startup GOOSY. The J11 is on a specific DECnet node, e.g. J11C:

```
$! file onl_tmr_startup.com
$ IF p1 .EQS. "" THEN INQUIRE p1 "Enter environment name "
$ IF p2 .EQS. "" THEN INQUIRE p2 "Enter node of J11      "
$ MDBM MOUNT BASE db db_j11.sec
$ DEFINE CAMAC_BRANCH_0 "''p2'::GESONE()" ! for ESONE commands
$ CRENV 'p1' /$TMR/$DSP/$DBM/$ANL
$ GOOSY INI ACQUIS NODE='p2' /J11
$ GOOSY LOA J11 j11c.cam          ! load the readout NAF list
$ GOOSY CLEAR SPECTRUM *          ! clear all spectra
$ EXIT
```

Similar command procedure to shutdown GOOSY:

```
$! file onl_tmr_shutdown.com
$ GOOSY STOP ACQUIS/ABO    ! break link to J11
$ DLENV                    ! delete environment
$ MDBM DISMOUNT BASE db
$ DEAS CAMAC_BRANCH_0
$ EXIT
```

Procedures Starting the Acquisition and Analysis

We write a small command procedure to start the acquisition:

```
$! file onl_tmr_start.com
$ GOOSY
START INPUT MAIL SIZE=8192  ! Size for J11 data 8K, MBD 4K
ATT DYN LIST accu           ! activate the Dynamic List 'accu'
SET ANAL/NOANAL             ! disable X$ANAL in the event loop
SET ACQUIS/SYNC/MAIL        ! Synchronize input with analysis
START ACQUIS
$ EXIT
```

Similar command procedure to stop the acquisition:

```
$! file onl_tmr_stop.com
$ GOOSY
STOP ACQUIS
STOP INPUT MAIL
DET DYN LIST accu
$ EXIT
```

Command Sequences

Now a typical session could proceed as follows:

```
$ @credb_j11                       ! create Data Base
$ @onl_tmr_startup <env> <node>    ! create GOOSY environment
$ @onl_tmr_start                   ! start GOOSY (dyn.list only)
$ GOOSY                            ! enter GOOSY prompter
GOOSY> SHOW ACQUIS
GOOSY> SHOW ANAL
GOOSY> TYPE EVENT 5             ! look if events are OK
GOOSY> ALL DEV <dev> PECAD/MAIN ! <dev> is the display terminal name
GOOSY> DISP PI raw
GOOSY> DISP SP energy
GOOSY> DISP SP time
GOOSY> DISP SP single(1)
GOOSY> REPLACE COND WIND peak   ! set condition by cursor
GOOSY> STOP ACQUIS
GOOSY> CLEAR SPECTRUM *         ! clear all spectra
GOOSY> DETACH DYN LIST accu     ! disable the Dynamic List 'accu'
GOOSY> SET ANAL/ANAL            ! enable X$ANAL
GOOSY> START ACQUIS
GOOSY> <CTRL>Z                  ! leave GOOSY prompter
$ @onl_tmr_stop
$ @onl_tmr_shutdown
```

6.14.5 Random Generator

The transport manager provides a random event generator. This generator is activated by the `/FOREIGN` qualifier:

```
GOOSY> INIT ACQUIS /FOR
```

There are 4 parameter events of type 4,1 generated.

6.14.6 Command Examples

The following examples cannot be taken literally but show the usage of some important commands. For more lucidity the parameters which have to be replaced by actual values are enclosed in brackets <>. We use the standard analysis program.

The first example is for a **J11** based single CAMAC create system.

```
$ CRENV <env> /$TMR/J11/$DBM         ! Create environment with standard
                                     ! analysis
$ GOOSY                              ! Enter GOOSY
GOOSY> INIT ACQUIS /J11 NODE=<node>  ! Initialize acquisition for J11
GOOSY> LOAD J11 j11c.cam             ! Load CAMAC table to J11
GOOSY> START ACQUISITION             ! Start data taking
```

The next example controls an **MBD** system using a private analysis program.

```
$ CRENV <env> /$TMR/$ANL/$DBM        ! Create environment with MGOOANL
$ GOOSY                              ! Enter GOOSY
GOOSY> INIT ACQUIS /MBD              ! Initialize acquisition for MBD
GOOSY> LOAD MBD GOO$IO:EXEC.BDO/EXEC ! Load exec code into the MBD
GOOSY> LOAD MBD GOO$IO:ESONE.BDO     ! Load ESONE code into the MBD
GOOSY> LOAD MBD <mycode>.BDO         ! Load user code into the MBD
GOOSY> LOAD STAR <s1>.EXE 1 23 /BOOT ! Load and boot J11 of crate one
GOOSY> LOAD STAR <s2>.EXE 2 23 /BOOT ! Load and boot J11 of crate two
GOOSY> START ACQUISITION             ! Start data taking
```

From here use the same commands for both examples:

```
GOOSY> SHOW ACQUIS                   ! Look what happens
GOOSY> STOP ACQUIS                   ! Stop data taking
GOOSY> START OUTPUT FILE <file.lmd>  ! Open file for output
GOOSY> START ACQUIS
GOOSY> STOP ACQUIS
GOOSY> STOP OUTPUT FILE /CLOSE       ! Close file

GOOSY> ATTACH DYNAMIC LIST accu      ! Activate the Dynamic List 'accu'
GOOSY> START INPUT MAILBOX SIZE=8192 ! Open input channel
                                      (8192 for J11, 4096 for MBD)
GOOSY> START ACQUIS
GOOSY> SHOW ANALYSIS                 ! View analysis
GOOSY> STOP ACQUIS                   ! Stop data taking
GOOSY> STOP INPUT MAILBOX            ! Stop analysis
GOOSY> DETACH DYNAMIC LIST accu      ! No longer execute the Dynamic List 'accu'
GOOSY> DETACH ANALYSIS               ! Free Data Bases for modifications

GOOSY> ATTACH ANALYSIS               ! Reinitialize analysis
GOOSY> ATTACH DYNAMIC LIST accu      ! Activate Dynamic List 'accu'
GOOSY> START INPUT MAILBOX SIZE=8192 ! Start analysis input
                                      (8192 for J11, 4096 for MBD)
GOOSY> START ACQUIS                  ! Start again
GOOSY> STOP ACQUIS
GOOSY> STOP INPUT MAILBOX
GOOSY> <CTRL>Z                       ! Leave GOOSY
$ DLENV                              ! Delete environment
```

Part III

GOOSY Data Management

Chapter 7

Data Management Introduction

7.1 Data Base Organization

In a software system for data acquisition and data analysis a large number of differently structured Data Elements has to be handled. Those Data Elements could be simple variables like calibration parameters or complex structures like a spectrum. Manipulations like create, delete, modify, copy, and show are required for those data objects. The Data Elements must be sharable in memory, i.e. several programs must access the same Data Element at the same time. Any Data Element must be accessible by programs and by commands. For example, a spectrum must be filled by an analysis program, displayed by a display program and shown by a command. Therefore the Data Elements are collected in Data Bases.

The **Data Bases** of GOOSY are implemented on VAX computers as structured Global Sections. The **Data Elements** are stored in sections of the Data Base called **Data Area**. Each Data Area is a cluster of contiguous pages which will be mapped into a program's address space. Data Areas with similar mapping attributes are collected in **Data Pools**. Information about Data Elements is kept in **Data Element Directories**.

The Data Base is organized internally in a hierarchical manner. Each component of the organization can be addressed by name. All names are collected in Directories, the Directory names in a Master Directory. The data regions are split in Data Areas, the Areas are bundled in Data Pools. The names of the Data Areas and the Data Pools are collected in the Area Directory and the Pool Directory, respectively. A Home Block keeps the entry information about the main Directories and the storage information for the Data Areas, i.e. the Data Base usage. Figure 7.1 on page 149 gives a simple overview about the Data Base Organization.

In the following, the objects of a Data Base are described briefly. The list begins with the smallest entity which can be referenced:

- **Member Value**

  This is the smallest entity, which can be **located** and **accessed** via the Data Management. A Member Value is a simple variable of a specific Data Type. The supported Data Types are:

  ```
  BIN FIXED(7)
  BIN FIXED(15)
  BIN FIXED(31)
  BIN FLOAT(24)
  BIN FLOAT(53)
  BIT(*)  ALIGNED
  CHARACTER(*)
  CHARACTER(*) VAR
  OFFSET
  UNKNOWN
  ```

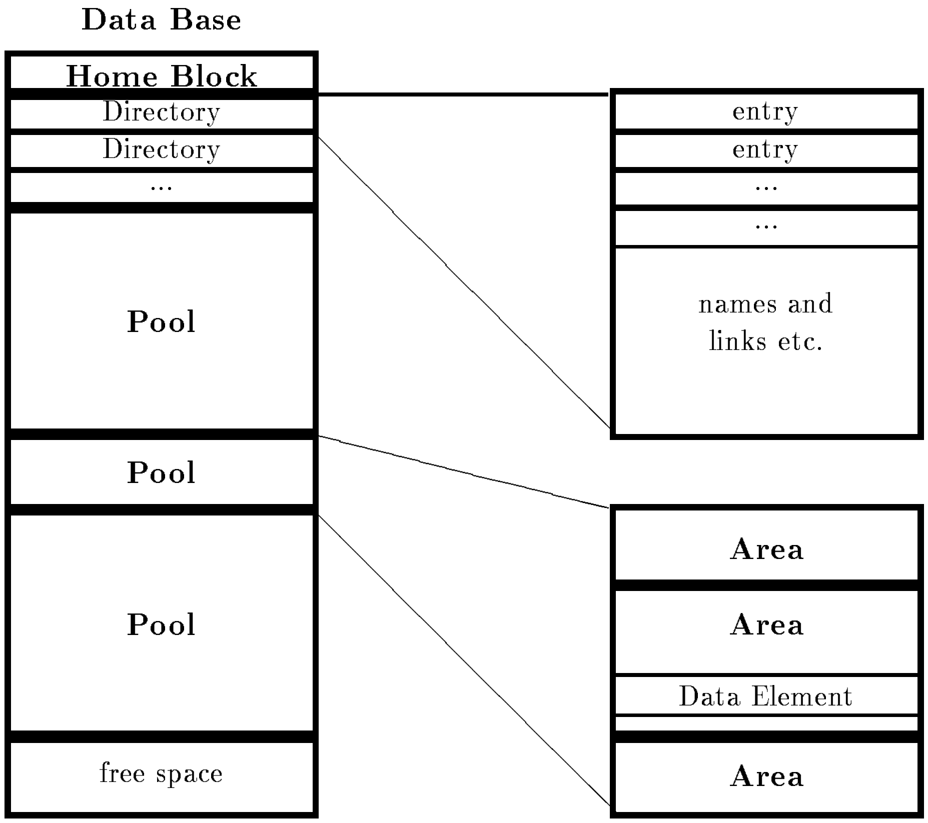


Figure 7.1: The simplified structure of a GOOSY Data Base.

- **Element Member**

  An Element Member is a Member Value or a 1 to 8 dimensional array of Member Values.

- **Data Element**

  A Data Element is the basic entity which can be manipulated by the Data Management. The structure of a Data Element is defined by a Data Type.

  The three basic Data Element forms are:

  - simple Data Element
    Containing only one Member Value or Element Member and corresponding to a simple variable or a simple array of PL/I.
  - complex Data Element
    Containing several Element Members and corresponding to a structure of PL/I.

  - indexed Data Element (name arrays)
    This is an array of Data Elements corresponding to a pointer array of PL/I referencing structures of the same Data Type.

- **Data Area**

  A Data Area is the smallest entity which can be made accessible (mapped) to a program. This is normally invisible to the user.

- **Data Pool**

  A Data Pool is the collection of Data Areas which requires the same specific combination of **protection** attributes. All Data Areas of a Data Pool are logically linked together. If a Data Area is short of space, a new Data Area can be created with the same attributes as the previous one. This new Data Area will be linked to the same Data Pool. A user normally knows the Data Pool only, not the Data Area.

- **Data Base**

  All information associated with one application is normally collected in one Data Base. A Data Base is a storage area, in which Data Pools can be created.
  Every Data Base has the following **protected system Data Areas**:

  Home Block: The Home Block is a specific part of the Data Base. The Home Block is always located at the begin of every Data Base and contains all information to **locate** other Data Areas, especially the **Directories**. It also keeps an allocation bit map of the whole Data Base and general informations like the section file name, creation date, and time etc.

  Area Directory: This Directory contains the relative addresses, the length in pages (512 bytes), the allocation cluster size, and the names of all Data Areas for the whole Data Base. It also keeps the Data Pool backward-link information for each Data Area.

  Pool Directory: This Directory contains the names of all Data Pools for the whole Data Base and the minimum size in bytes of any Data Area in this Data Pool. It also keeps the link information for the first Data Area of each Data Pool.

  Master Directory: This Directory contains the names of all Data Element Directories and the relative addresses of the Directory Areas in the Data Base.

  Data Element Directories: Each of these Directories contains the names of all Data Elements, the relative addresses of their Data Areas, the relative addresses within the Data Areas, and other Data Element information for all Data Elements of one Data Element Directory. This information is called **Directory Entry**.

  There are three additional extensions possible for each entry in a Data Element Directory:

  - Any extension of the Data Element Descriptor. These extensions may be of any length. They are characterized by an extension type. All extensions of a Data

Element Descriptor are linked. The character string of the name of a Data Element is located within such an extension.

– A queue of Data Elements of the same Data Element Directory, i.e. in the same Data Element Directory. This allows to bind unnamed Data Elements to named Data Elements, e.g. a named spectrum header and its unnamed spectrum limit definitions.

– A link to Data Elements of any Data Element Directory. This allows logical correlations of Data Elements, e.g. conditions linked to a spectrum.

Type Directory: The Type Directory is a specific Data Element Directory which contains the Data Types declarations of all Data Elements in the Data Base. Each new Data Type must be inserted in the Type Directory before it can be used to create a Data Element.

7.2 Glossary

Data Base A formatted VMS global section. The Data Base name is a logical name of a VMS global section or the global section name itself.

Global Section Part of memory which can be shared by several processes. Provided by VMS.

Global Section File Each global section is created as a file. Parts of the section can be mapped into a process virtual address space. Global section pages are paged to the global section file.

Data Base Area Contiguous number of pages in the Data Base (global section file).

Data Base Pool Composed of several Areas. Smallest entity which can be mapped by a process. One Pool has for one process one access mode (Write or read only). If a Pool runs out of space, one more Area is chained to that Pool.

Data Element Piece of data in the Data Base. It has a name which is kept with other information in a Directory. The data part is kept in a Pool (Area). The structure of a Data Element is described by a PL/I structure declaration.

Data Element Member Member of a Data Element structure.

Data Element Array Data Elements can be indexed in up to two dimensions. Each element of such an array has its own slot in the Directory. Therefore the data structures could be different.

Data Element Type A Data Element describing a PL/I structure. It is created from a PL/I structure declaration. Data Elements are created with that data structure by referring to the corresponding Type.

Data Element Directory Information about Data Elements, Areas, Pools, Data Types and Directories is kept in Directories.

Mount/Dismount Mount a Data Base means to create a global section. The global section file (Data Base) must exist.

Attach Attach a Data Base, Pool or Directory means to map the appropriate parts of the Data Base into the memory of the attaching process. The resulting pointers are kept in a local mapping context structure normally invisible to the user. This operation is needed for programmed access to Data Elements.

Locate Locate a Data Base, Pool, Directory or Data Element means to get the pointer to the object and/or get an identification number. This number is valid and unique during the lifetime of the object. It can be used for a fast attach or locate to get the pointer. This operation is needed for programmed access to Data Elements.

All these functions can be done by commands executed by the Data Base Manager. The commands are described in the following chapters.

7.3 GOOSY Command Interface

7.3.1 Line Input

The GOOSY command syntax is similar to the DCL syntax. A command is composed of several **keywords**. **Positional parameters are delimited by spaces**. In addition to DCL, however, they have names and may be specified in any order by `name=value`. The names are found in the help description of the command or in the menu. **Qualifiers** are preceded by a slash (`/qualifier`). Different to DCL they cannot specify a value. Qualifiers can in most cases be negated by `/NOqualifier`. Qualifier sets are mutually exclusive qualifiers. In the menu display the positional parameters are specified on top together with their names, followed by the qualifiers. At last, the qualifier sets follow. An example is shown below.

7.3.2 Command Procedures

The GOOSY command interface is able to process command files. These files must contain GOOSY commands lines. The default file type is .GCOM. The commands in these files are executed by the '@' command:

```
   DBM> @file
   GOOSY> @file
```

Comment lines may be written starting the line with an exclamation point (!). Lines can be continued by ending up with an hyphen (-). Arguments can be passed to the procedure. These arguments replace in the procedure text the strings &P1 (first argument) ... &P8. Example: Procedure CS.GCOM contains a command to create a spectrum:

```
CREATE SPECTRUM &P1 L &P2 /DIG
```

This procedure is called by

```
   DBM> @CS SP1 0,4096
```

7.3.3 Defaults

In some cases default values are provided for command parameters. Some defaults are replaced by the input values, if specified. These parameters are called **replaceable**. **Global parameters** are valid for several commands, e.g. the Data Base name. Global parameters can be preset in the profiles. Some parameters are **required**, i.e. they are prompted if not specified. In the command descriptions these attributes are described for each parameter.

7.3.4 Conventions in the Data Base Manager

Defaults

Most of the commands of the Data Base Manager concern Data Elements. The full specification of a Data Element would be:

```
  node::base:[directory]name(index)
for example
   E::DB:[DATA]PAR(5)
   DBM> SHO SPEC MVIIA::TEST:[$SPECTRUM]S5
```

(Remote access is not yet implemented!) Data Element Members have to be specified as

```
   node::base:[directory]name(index).member
for example:
   DBM> SHO MEMBER E::DB:[DATA]PAR(5).CAL(3).ENER
```

In this example `PAR` is an indexed Data Element. To provide a more convenient specification the names for the node, base and Directory can be specified by separate parameters. These parameters are replaced by the specified values and thus defaulted in subsequent commands, as shown in the following command sequence:

```
   DBM> SHO SPEC s1 SPEC_DIR=test BASE=db
   DBM> SHO SPEC s2        ! uses Directory 'test' in Data Base 'db'
   DBM> SHO ELEM event DIR=data BASE=db
   DBM> SHO ELEM newevent  ! uses Directory 'data' in Data Base 'db'
   DBM> SHO ELEM [para]p1  ! uses Directory 'para' in Data Base 'db'
   DBM> SHO ELEM newevent  ! uses again Directory 'data' in Data Base 'db'
```

If these replaceable parameters are in addition global, i.e. valid for more then one command, they are preset in **profiles**. The profile is accessed by a logical name GOO$PROFILE. The standard GOOSY profile is GOO$EXE:PROFILE.PROF. It sets the following parameter values for the Data Base manager:

```
     BASE=DB
     BASEFILE=DB
     AREA=
     POOL=DATA
     DIR=DATA
     DYN_LIST=L1
     DYN_TYPE=SPECTRUM
     NODE=*
     NAME=
     LINK_FROM=
     LINK_TO=
     DYN_DIR=$DYNAMIC
     COND_DIR=$CONDITION
     SPEC_DIR=$SPECTRUM
     PIC_DIR=$PICTURE
     POLY_DIR=$POLYGON
     CAL_DIR=$CALIB
     DYN_POOL=$DYNAMIC
     COND_POOL=$COND_POOL
     SPEC_POOL=$SPEC_POOL
     PIC_POOL=$PIC_POOL
     POLY_POOL=$PIC_POOL
     CAL_POOL=$CAL_POOL
     PAR_DIR=DATA
```

These initial values can be reset by the commands

```
    DBM> $ RESET DEFAULT
    GOOSY>$DBM>$ RESET DEFAULT
```

depending if the Data Base Manager was started stand alone or as component in a GOOSY environment.

Array Values

Specification of **value arrays** for a parameter is done as a list of values separated by commas and optionally enclosed in parenthesis. E.g. spectrum limits:

```
    DBM> CRE SPEC s L (0,1023,0,1023) BINS=4,8
```

Wildcards

Many commands, especially SHOW commands, provide wildcard facilities. Data Element names can be wildcarded with asterisks:

```
     *   all names
     x*  all names starting with string 'x'
     *x  all names ending with string 'x'
     x*y all names starting with string 'x' and ending with string 'y'
     *x* all names containing string 'x'
```

The asterisk matches any string inclusive null string.

7.3.5 Command Example

The command CREATE SPECTRUM has several positional parameters and qualifiers.

```
CREATE SPECTRUM name type limits binsize
                /[NO]ERRV
                /[NO]ERRH
                /ANALOG/DIGITAL
```

The following are valid commands:

```
CRE SPEC s1 L (0,1000) 1 /DIG
CRE SPEC s1(5) L (0,1000) 1 /DIG SPEC_DIR=test
CRE SPEC [test]s1 L (0,1000,0,1000) 1 /DIG
CRE SPEC /DIG s1 L 0,1000 1
CRE SPEC s2 LIM=0,100 TYPE=R /ANA /ERRV
```

Some **invalid** commands:

```
CRE SPEC LIM=0,100 s1   ! invalid positional after named reference
CRE SPEC s1 LIM=(0 100) ! invalid space invalid delimiter in lists
CRE SPEC s1 /ANA/DIG    ! invalid exclusive qualifiers
```

7.3.6 GOOSY Menu

GOOSY menus are specific to GOOSY components. The command MENU enters the GOOSY component menu. Some keys are defined for a shorter menu access: As input to a GOOSY component prompt, press $\boxed{\texttt{NEXT SCREEN}}$. Example:

```
MDBM                        ! start Data Base Manager
SUC: DBM> <NEXT SCREEN>     ! pressing this key enters the menu.
```

There are two types of menus, one for commands and one for the command parameters which is entered when you reach a full command (commands may be composed of several keywords like in DCL). The command menu looks like

```
keyword
Subcommands available ================================================
keyword   * :  * :list of available subcommands
keyword     : short description
************** End of list ******************* End of list ***
```

The first line type is for a command which needs more subcommands, the second is the layout of an executable command. Entering a keyword of the first type, the next menu command level is displayed. Entering a keyword of the second type, the parameter menu of this command is displayed. Several keywords may be entered at once. The parameter menu looks like:

```
keyword keyword...
short description
Positional parameter list =====================================
parameter name |type|short description      =default
parameter name |type|short description      :default
Qualifier list -------------------------------------------
qualifier      |SWI |short description      :default
Qualifier set list ---------------------------------------
set name       |SET|list of possible values  :default
********* End of list ****************** End of list ********
```

In this menu one moves the cursor around using the arrow keys and can overwrite the displayed default values. Note that qualifiers always must be preceeded by a slash (`/qualifier`). An $\boxed{=}$ sign before the default marks required parameters. Press the $\boxed{\text{KP\_PF2}}$ key to get a keypad layout. Press the $\boxed{\text{HELP}}$ key or $\boxed{\text{KP\_PF1}}$ and $\boxed{\text{KP\_PF2}}$ to get help information for the present level. All defaults presently active are displayed in the menu. If the command is executed (by $\boxed{\text{RETURN}}$), the actual command line is displayed on top of the screen. You leave the component or prompter or menu by (several) $\boxed{\text{CTRL}}$Z. To return to previous command level without executing the current menu command, press $\boxed{\text{KP\_PF4}}$.

7.3.7 ALIAS Names

GOOSY commands including parameter specifications can be replaced by alias names. These names are defined on two levels: global and environment. The environment level names are searched first. They are activated by the `CRENVIR` command and deactivated by the `DLENVIR` command. If no environment is active, only the global alias names are valid. Alias names cannot be abbreviated. They are implemented on GOOSY command level and DCL command level (DCL symbols). All alias names are deleted during logout. Therefore it is recommended to create alias names in the LOGIN.COM procedure using the DCL command `ALIAS CREATE`.

DCL Commands

DCL commands to handle alias names:

```
$ ALIAS CREATE name string environment /GLOBAL
$ ALIAS DELETE name environment /GLOBAL
$ ALIAS SHOW name environment /GLOBAL/ACTIVE
```

To create global alias names, omit the environment or use the `/GLOBAL` qualifier. Examples:

```
$ ALIAS CRE ANA "CREATE PROC MGOOANL"      !global alias ANA is created
$ ALIAS CRE INAC "INI ACQ /J11 NODE=" SUSI !envir.alias INAC is created
$ ALIAS SHO ENV=SUSI                       !envir.alias names of SUSI
$ ALIAS SHO                                !All active alias names
$ ALIAS SHO /GLOB                           !Global alias names only
$ ALIAS SHO INAC                            !Alias INAC
```

GOOSY Commands

Sometimes it is useful to create alias names from GOOSY command level. GOOSY commands to create, delete and show alias names are implemented in the GOOSY prompter, in the standalone Data Base Manager and the stand alone display. The arguments are the same as shown above, but the commands begin with the verb to fit into the GOOSY commands.:

```
DBM> CREATE ALIAS name "string"
GOOSY> SHOW ALIAS
DISP> DELETE ALIAS name
```

Alias names created by GOOSY commands are not defined as DCL symbols.

Chapter 8

Data Base Manager

8.1 Data Base Manager Program

The Data Base Manager is invoked stand alone by the DCL command:

```
   $ MDBM                       ! start DBM
   SUC:DBM> <NEXT SCREEN>       ! pressing this key enters the menu.
```

A Data Base should have been created already, e.g. by CREDB.

8.1.1 Data Base Manager Menu

The top menu level looks like:

```
Subcommands available ==================================================
$ *          :  * :ATTACH,CALL,DCL,DEBUG,DEFINE,DIRECTORY,EXECUTE,EXIT,..
CALCULATE * :  * :SPECTRUM
CALIBRATE * :  * :SPECTRUM
CLEAR *      :  * :CAMAC,CONDITION,ELEMENT,PICTURE,SPECTRUM
COPY *       :  * :CONDITION,ELEMENT,MEMBER,POLYGON,SPECTRUM
CREATE *     :  * :ALIAS,AREA,BASE,CALIBRATION,CONDITION,DIRECTORY,...
DECALIBRATE :  * :SPECTRUM
DELETE *     :  * :ALIAS,CONDITION,DYNAMIC,ELEMENT,LINK,POOL,..
DISMOUNT *  :  * :BASE
DUMP *       :  * :SPECTRUM
FREEZE *     :  * :CONDITION,SPECTRUM
HELP         : Access VMS HELP facility
LOCATE *     :  * :BASE,DIRECTORY,ELEMENT,ID,POOL,QUEUEELEMENT,TYPE
MENU         : Enter menu
MODIFY *     :  * :DIRECTORY,FRAME,TABLE
MOUNT *      :  * :BASE
READ *       :  * :CAMAC
SET *        :  * :CONDITION,LETTERING,MEMBER,SPECTRUM
SHOW *       :  * :ALIAS,AREA,CALIBRATION,CAMAC,CONDITION,DIRECTORY,...
START *      :  * :MR2000
STOP *       :  * :MR2000
SUMUP *      :  * :SPECTRUM
UNFREEZE *  :  * :CONDITION,SPECTRUM
UPDATE *     :  * :BASE,DYNAMIC
WRITE *      :  * :CAMAC
************** End of list ******************* End of list ***
Command:
 PF2: Help, PF3: Enter command, PF4: Break, ENTER: Previous menu
Subcommand :
```

8.2 Data Base Manager Component

8.2.1 The GOOSY Environment

As shown above, the Data Base Manager may be executed standalone on DCL level. It may, however, also be bundled together with other GOOSY programs in an **environment**. In an environment several components are started together. Commands to be executed by environment components are dispatched by the GOOSY prompter from one terminal. Therefore, on each terminal one has to create an environment. Commands given from that terminal are executed by the components running in that environment. To create an environment with optional components, use the DCL command `CRENVIR`. The Data Base Manager is started by the qualifiers `/ONLINE` or `/OFFLINE` or `/$DBM`.

```
$ CRENVIR ?                       ! Enter menu
$ CRENVIR environment /ONLINE     ! All GOOSY components
$ CRENVIR environment /OFFLINE    ! All except Transport Manager
$ CRENVIR environment /$DBM       ! Data Base Manager only
```

The environment name must be unique within a user group on one VAX node. It can be one to four characters long.
Similar, the DCL command

```
$ DLENVIR
```

deletes the present environment including all components (subprocesses).

8.2.2 The GOOSY Prompter

The GOOSY prompter is entered by the DCL command `GOOSY`:

```
$ GOOSY
SUC:GOOSY>
```

Now you can enter any GOOSY command. You leave the GOOSY prompter by typing $\boxed{\texttt{CTRL}}$ `Z`. Single GOOSY commands can be given under DCL by a preceding `GOOSY` or just `G`. This allows to execute all GOOSY commands as DCL commands and to replace DCL symbols in the GOOSY command line.

The GOOSY prompter interprets some special function keys. The definitions are made in GOO$EXE: INI_TP0.COM. A help setup is displayed by the $\boxed{\texttt{KP\_PF2}}$ key.

Now, entering the GOOSY prompter, the menu for the Data Manager component is entered by:

```
<KP_MINUS> ! $DBM : Data Base Manager commands menu
```

Similar, the Data Manager component can be deleted and created by special keys:

```
   <PF1><KP_COMMA>  ! delete $DBM : Data Base Manager
   <KP_COMMA>       ! create $DBM : Data Base Manager
```

The Data Base Manager is not necessary needed to acquire or analyze data. It may be restarted at any time without affecting other GOOSY components in the environment.

Chapter 9

Data Management Commands

9.1 Data Base Commands

In the following some commands used for general purpose are discussed. The commands can be given to the DBM> prompt or to the GOOSY> prompt if an environment with $DBM component is created. Note that in the following descriptions lower case names have to be replaced by meaningfull values. Uppercase names are keywords and must be typed as shown.

9.1.1 Data Base

Any Data Base must be created once. During the creation of a Data Base the Data Base file is created. In addition, a Data Base is a VMS global section. Therefore the file must be made a global section file. This is called **mounting** the Data Base. The reverse is to **dismount** the Data Base. This means that the file is no longer a global section file, i.e. it can be deleted or copied. The Data Base must be mounted before any of the commands described in the following is executed. It can be mounted only on one VMS node, because a Data Base can be shared in memory by several programs. The Data Base file must be accessible on the node where it is mounted. In a VAX cluster that means that the Data Base must be dismounted on one node before it can be mounted on another node. The file is accessible from all cluster nodes and needs not to be copied. All Data Bases remain mounted until they are dismounted or the VAX operating system is rebooted.

Create/Delete a Data Base

To create a Data Base one may use the command

```
DBM> CREATE BASE base basefile ade mde pde tde basepages
```

After that the Data Base is mounted. It is recommended to give the file the same name as the Data Base with the file type .SEC. For the Data Base name a VMS logical name may be created and used in all other commands. The four parameters `ade, mde, pde, tde` specify the number of Areas, Directories, Pools and Data Types which can be created in the Data Base. The parameter basepages specifies the size of the Data Base in pages (512 bytes). These parameters cannot be extended later. A DCL command is provided to create GOOSY Data Bases:

```
$ CREDB ?            ! Enter menu
$ CREDB basename filename size[KB]
       /SPECTRA=s    ! maximum number of spectra
       /PICTURE=p     ! maximum number of picture frames
       /CONDITIONS=c  ! maximum number of conditions
       /DIRECTORIES=d ! maximum number of Directories
       /POOLS=p       ! maximum number of Pools
       /POLYGON=p     ! maximum number of polygons
       /DYNLIST=d     ! maximum number of Dynamic Lists
```

This command creates the Directories $SPECTRUM, $CONDITION, $PICTURE, $POLYGON, $DYNAMIC, DATA and the Pools $SPEC_POOL, $COND_POOL, $PIC_POOL, $DYNAMIC and DATA. The command procedure creating the GOOSY Data Base can be saved by qualifier `/SAVE=file`. This file may be edited and used later instead of `CREDB`.

Mount/Dismount a Data Base

When a Data Base has been created, it is mounted. A Data Base is dismounted by

```
   DBM> DISMOUNT BASE base
or in DCL by
   $ DELGS name
```

After that the Data Base file can be copied, deleted or dumped, but the Data Base cannot be accessed by commands. To mount the Data Base again, use

```
   DBM> MOUNT BASE base basefile
```

Compress/Expand a Data Base

If a Data Base is not used for some time, it can be compressed to save disk space. The compressed Data Base file contains no zeros. Before a Data Base can be mounted on this file, the file must be expanded.

```
    $ MUTIL COMPRESS BASE base basefile file
```

After that the Data Base is dismounted except the `/MOUNT` qualifier was given. The file 'file' must not exist. To expand a compressed Data Base file use

```
    $ MUTIL DECOMPRESS BASE file base basefile
```

The file 'basefile' must not exist. The Data Base 'base' must not be mounted. After that command the Data Base is not mounted except the `/MOUNT` qualifier was given. The default file type for compressed files is .CSEC.

9.1.2 Data Base Pool

The data part of the Data Elements is allocated in Pools. Pools are composed of chained Areas. Areas are sections of contiguous pages of the Data Base. The Pool size is no limit of the Pool, because it is extended automatically by adding one more Area to the Pool. One should at least specify the size of the largest Data Element to be allocated in the Pool.

```
   DBM> CREATE POOL pool size              ! size in bytes
```

If the Pool size is chosen too small for the Data Elements to be allocated in the Pool, one Area is created for each Data Element. This costs entries in the Area Directory and time to map the Pool. If the size is too big, it wastes space in the Data Base. The command `SHOW POOL` shows a list of existing Pools, the Areas of a Pool and the usage of the Areas.

9.1.3 Data Base Area

As mentioned above, Areas are parts of Pools. The user is normally not involved with Areas. Areas are created automatically. But if Pools are chosen too small, many Areas are created and the Area Directory may be too small. To check that, the command

```
DBM> SHO AREA * /DIR
```

shows a list of all Areas created and

```
DBM> SHO AREA area /FULL
```

shows the usage of an Area.

9.1.4 Data Base Directory

Each Data Element in the Data Base uses one entry slot in a Directory. The information kept in the Directory is used to access the Data Element. The name of the Data Element, the Pool where the data part is allocated and links to other Data Elements are kept in the Directory entry slot.

```
DBM> CREATE DIRECTORY directory entries
```

Creating Directories one should know that each member of a Data Element name array takes one entry in the Directory. Some GOOSY Data Elements take more than one entry, i.e. spectra four, conditions two, composed conditions three and pictures one per picture plus one per frame. The command `SHOW DIRECTORY` shows a list of all Directories and the content and usage of a Directory.

Expand a Data Base Directory

If a Directory has no more free slots, it can be expanded by

```
DBM> MODIFY DIRECTORY directory entries
```

It may happen that there is no more space in the Directory even if there are still free entries. This can be caused by very many links. The columns 'Used bytes for links' and 'Free bytes' in the output of `SHOW DIRECTORY /FULL` show this situation. If there are less than 100 bytes free, the Directory should be modified to a bigger size. A Directory can be deleted only if there are no used entries in it.

9.1.5 Data Element Declarations

Before creating a Data Element, one must specify the structure declaration. This is done by a PL/I structure declaration. This declaration must be in a file or text library module. The name of the file or library module must be the name of the structure. It must made known to the Data Base. This is done by `CREATE TYPE`:

```
DBM> CREATE TYPE @library(module) ! Declaration from library
DBM> CREATE TYPE @filename        ! Declaration from file
```

The same file or library module may be included in a PL/I program to access the Data Element. Note that the structure should be declared as BASED. The default base pointer may be declared in the same file or library module. In PL/I programs the pointer to the Data Element is returned by a Data Management routine or macro. A Type declaration can be deleted only if there are no Data Elements referring to that type. The Type declarations known can be obtained by

```
DBM> SHOW DIRECTORY $TYPE /FULL
```

Note the default declarations for 'simple' Data Elements:

```
H  for  BIN FIXED(7)
I  for  BIN FIXED(15)
L  for  BIN FIXED(31)
R  for  BIN FLOAT(24)
D  for  BIN FLOAT(53)
C  for  CHARACTER
V  for  CHARACTER VAR
B  for  BIT ALIGNED
```

These Types may be used to create simple Data Elements composed of one member or array. A specific declaration can be output by

```
DBM> SHOW TYPE type
```

where 'type' is the name of the structure.

9.1.6 Data Elements

Create Data Elements

There are two kinds of Data Elements: **simple** and **complex**. Simple Data Elements are like non structured PL/I variables or arrays. Complex Data Elements are like PL/I structures. Simple Data Elements can be created without Type declarations. The Type parameter specifies the Data Type (H,I,L,R,D,C,V,B) and optionally the array dimensions:

```
DBM> CREATE ELEMENT name pool DIR=directory TYPE=datatype
DBM> CREATE ELEMENT e1 POOL=data DIR=data TYPE=L
DBM> CREATE ELEMENT e2 POOL=data DIR=data TYPE=L(16)
```

Note that Directory and Pool must exist. Complex Data Elements are described by Type declarations which must be created in the Data Base. The declarations used in the CREATE ELEMENT command must not contain REFER values.

```
    DBM> CREATE TYPE @library(struc)       ! Create Type declaration
    DBM> CREATE ELEMENT e1 POOL=data DIR=data TYPE=struc
```

The space allocated in the Pools by Data Elements is controlled by bitmaps. Each bit in the bitmap represents a contiguous number of bytes. This number can be specified together with the `CREATE ELEMENT` command.

```
    DBM> CREATE ELEMENT ... CLUSTER=bytes
```

This number should be a fraction of the smallest Data Element allocated in the same Pool. If the number is too big, that means that space is wasted because it is the smallest entity allocatable in the Pool. If it is too small space is wasted because of the size of the bitmap. The number need not to be the same for all Data Elements in the Pool. Only Data Elements with the same clustersize, however, can be allocated in the same Area, i.e. there are at least as many Areas in a Pool as Data Elements with different cluster sizes.

Data Element Name Arrays

Data Elements can be indexed, i.e. to build arrays of Data Elements. The index is added to the Data Element name.

```
    DBM> CREATE ELEMENT ea1(20) TYPE=x ...
```

creates 20 Data Elements 'ea1' of Type 'x' using 20 entries in the Directory. All array members have the identical Data Type and Pool during the creation of the Data Element array. To vary single array members use the `CREATE` command with the `/REPLACE` option after the whole Data Element name array was created, e.g.

```
    DBM> CREATE ELEMENT ea1(3) POOL=data TYPE=type /REPLACE
```

A single element of such a Data Element name array can be referenced in SHOW commands, but not in DELETE commands. One can only delete the whole data element name array.

Delete Data Elements

Data Elements can be deleted if there are no links on the Data Element (see below) and if no program has access to the Data Element. In the second case an error message will be displayed telling that there are 'locks' for that Data Element. Specifying the name of a Data Element name array the whole array is deleted.

```
    DBM> DELETE ELEMENT name DIR=directory
    DBM> DELETE ELEMENT [directory]name
```

When the Directory is specified as a separate parameter it is used in subsequent commands as default. When specified as part of the name it is used only temporary.

Data Element Links

Data Element links are used by the data management to protect Data Elements referring each other. In the future they may be used to execute commands for arbitrary groups of Data Elements linked together. A link is always directed from one Data Element to another Data Element which could be the same. As long as there are outgoing or incoming links a Data Element cannot be deleted. The user is normally not involved in the link management. The `SHOW LINKS` command can be used to see the links of a Data Element. One never should delete links if one does not understand very well the situation. If Data Elements cannot be deleted because of links there is normally a reason for that.

9.1.7 Data Base Usage

As mentioned in the previous sections there are many `SHOW` commands to get information about the usage of a Data Base. These are discussed in more detail in the following sections.

SHOW HOMEBLOCK

All pages in the Data Base are controlled by a main bitmap. For each page allocated by an Area one bit in the bitmap is set. Thus freed pages can be used again and the total usage and fragmentation of the Data Base can be determined at any time. The command

```
DBM> SHOW HOMEBLOCK
```

shows the overall usage of the Data Base.

SHOW AREA

Similar the usage of each Area is controlled by Area bitmaps. A list of all Areas and detailed information about an Area is obtained by

```
DBM> SHOW AREA * /DIR    ! Usage of Area Directory
DBM> SHOW AREA *         ! Usage of known Areas
DBM> SHOW AREA xyz /FULL ! Show information and bitmap of Area xyz
```

SHOW POOL

This command outputs a list of all known Pools or detailed information about one specific Pool

```
DBM> SHOW POOL /DIR      ! Usage of Pool Directory
DBM> SHOW POOL /DIR /FULL! Usage of Pool Directory with all entries
DBM> SHOW POOL *         ! Usage of known Pools
DBM> SHOW POOL xyz       ! Show Areas of Pool 'xyz'
DBM> SHOW POOL xyz /FULL ! Show full information about
                         !  all Areas of Pool 'xyz'
```

SHOW DIRECTORY

This command outputs a list of all known Directories or detailed information about one Directory:

```
DBM> SHOW DIRECTORY *         ! list of known Directories
DBM> SHOW DIRECTORY xyz /FULL ! Show entries of Directory xyz
```

The entries of a Directory may show Data Elements without names (name=***). These Data Elements are queued to a master Data Element (Queue Header). They can be accessed only via their queue header. All GOOSY Data Elements like spectra and conditions use this mechanism. A spectrum is composed of three Data Elements queued to the spectrum header Data Element. The data part of queued Data Elements can be allocated in different Pools. Thus the different parts of a spectrum could be accessed in a program by different access modes, i.e. read only or read/write.

Another feature can be seen by this command. Data Elements can be created as name arrays, i.e. the Data Element name is indexed. For each element of such an array one entry in the Directory is used.

The order in the output of the `SHOW DIRECTORY` command is not alphabetically. To get an alphabetic order one should use the command `SHOW ELEMENT`.

SHOW ELEMENT

Any Data Element, regardless of its structure (Type), can be displayed. The information stored in the Directory entry or the information stored in the data part can be selected:

```
DBM> SHOW ELEMENT name
DBM> SHOW ELEMENT [directory]* ! Data Element list
DBM> SHOW ELEMENT name /FULL   ! Directory information
DBM> SHOW ELEMENT name /DATA   ! Data part of the Data Element
```

SHOW MEMBER

Similar to PL/I the smallest entity of a structure is called a Member. A Data Element Member can be set to a value and the current value can be output. This is done by the commands:

```
DBM> SET  MEMBER [directory]elementname.member value
DBM> SHOW MEMBER [directory]elementname.member
```

The expression 'member' may be a list of several Members separated by periods depending on the structure.

Chapter 10

GOOSY Data Elements

There are some GOOSY Data Elements which are handled by special commands. These are described in the next sections.

10.1 Conditions

By default, conditions are kept in the Directory $CONDITION. GOOSY conditions are independent of spectra or coordinates (parameters). All kinds of conditions can be executed in a Dynamic List. In an analysis routine they are executed by the macro $COND. They may then be used as filters for spectrum accumulation and/or scatter plots. Each condition has TRUE and FALSE counters and freeze, result, and preset bits.

10.1.1 Related Commands

```
CREATE   CONDITION type
DELETE   CONDITION
SHOW     CONDITION
CLEAR    CONDITION COUNTER
COPY     CONDITION
SET      CONDITION type
FREEZE   CONDITION
UNFREEZE CONDITION
```

Each condition takes two entries in Directory $CONDITION, except composed conditions which take three. The first Data Element is a condition header common to all conditions, the second keeps specific information. The default Pool is $COND_POOL.

10.1.2 Condition Header Data Element

The declaration of the header is kept in library GOOTYP(SE$COHE).

```
/* ----------------- start of SE$COHE ---------------------------*/
DCL 1 SE$COHE BASED,                        /* condition header  */
      2 BE$COHE_ATTR BIT(32) ALIGNED,       /* flags             */
      2 CVE$COHE_NAME CHAR(30) VAR ,        /* name redundant    */
      2 CE$COHE_TYPE CHAR(1),               /* summarizes Type   */
      2 CVE$COHE_DATE_TIME_CRE CHAR(23) VAR,/* creation date     */
      2 CVE$COHE_CREATOR CHAR(63) VAR,      /* Creating program  */
      2 LE$COHE_COTAB_DIR_ID BIN FIXED(31), /* table Directory   */
      2 LE$COHE_COTAB_DE_ID BIN FIXED(31),  /* table index       */
      2 LE$COHE_ID_IN_COTAB BIN FIXED(31);  /* cond.table index  */
/* ------------------- end of SE$COHE --------------------------*/
```

The different kinds of conditions are:

10.1.3 Window Conditions

Data Elements

Window conditions are implemented in two Data Elements. The declarations are kept in library GOOTYP(SE$COHE) and GOOTYP(SE$COWI).

```
/* ----------------- start of SE$COWI ---------------------------*/
DCL 1 SE$COWI BASED,                        /* window condition  */
      2 LE$COWI_TRUE_CT BIN FIXED(31) ,     /* true count        */
      2 LE$COWI_FALSE_CT BIN FIXED(31) ,    /* test count        */
      2 LE$COWI_DIM BIN FIXED(31) ,         /* dimension         */
      2 RE$COWI_LIMITS_LOW(1 REFER (LE$COWI_DIM)) BIN FLOAT(24),
    /* limits           */
      2 RE$COWI_LIMITS_UP(1 REFER (LE$COWI_DIM)) BIN FLOAT(24);
/* ------------------- end of SE$COWI --------------------------*/
```

A window condition keeps n window limits (subwindows) forming an n-dimensional cube. Points inside the cube are true, outside false. Up to eight subwindows may be used in a Dynamic List, up to four in macro $COND. Each limit pair may be applied to a different object. Object may be any Member of a Data Element which is a BIN FLOAT(24), BIN FIXED(31) or BIN FIXED(15) number. The condition is TRUE if all subwindows are TRUE.

Create Window Conditions

```
   DBM> CRE COND WINDOW c (1,1000) 1         ! 1 subwindow
   DBM> CRE COND WINDOW c (1,100,1,200)      ! 2 subwindows
   DBM> CRE COND WINDOW c (1,1000) 2         ! 2 subwindows
                                             ! both (1,1000)
   DBM> CRE COND WINDOW c(10) (1,100)        ! 10 cond., 1 subw. each
```

Set Window Condition Limits

In the Data Base Manager the limits of window conditions can be set only by values. To specify the limits by graphic input, use the command `REPLACE CONDITION WINDOW` which is executed in the display program.

```
DBM> SET CONDITION WINDOW condition limits dimension
DBM> SET COND WINDOW c (1,1000) 1          ! 1. subwindow
DBM> SET COND WINDOW c 1,100 2             ! 2. subwindows
```

10.1.4 Multiwindow Conditions

Multiwindows are composed of the same Data Elements as normal windows. The difference to normal window conditions is that there is one object for all subwindows and one result bit for each subwindow. In a Dynamic List or macro any number of subwindows is processed. The result bits can be used as filters for spectrum array accumulation. The number of the last TRUE subwindow may be used to select a spectrum array member for accumulation (See MULTIWINDOW and INDEXEDSPECTRUM in Dynamic Lists). In macro $COND two execution modes can be selected: $COND(MW,...) executes all subwindows returning all bits and the index of the last true subwindow. $COND(MWI,...) stops execution after the first true subwindow check. If only the index is needed this is a faster mode.

Create Multiwindow Conditions

```
DBM> CRE COND MULTI  c (1,1000) 100        ! 100 subwindows
```

Set Multiwindow Condition Limits

In the Data Base Manager the limits of window conditions can be set only by values. To specify the limits by graphic input, use the command `REPLACE CONDITION WINDOW` which is executed in the display program.

```
DBM> SET CONDITION WINDOW condition limits dimension
DBM> SET COND WINDOW c (1,1000) 1          ! 1. subwindow
DBM> SET COND WINDOW c 1,100 2             ! 2. subwindows
```

10.1.5 Pattern Conditions

Data Elements

Pattern conditions are implemented in two Data Elements. The declarations are kept in library GOOTYP(SE$COHE) and GOOTYP(SE$COPA).

```
/* ------------ start of SE$COPA --------------------------------- */
DCL 1 SE$COPA BASED ,                      /* pattern condition     */
      2 LE$COPA_TRUE_CT BIN FIXED(31) ,     /* true count            */
      2 LE$COPA_FALSE_CT BIN FIXED(31) ,    /* test count            */
      2 BE$COPA_MODE BIT(16) ALIGNED,       /* Mode                  */
      2 LE$COPA_DIM  BIN FIXED(31),         /* dimension             */
      2 SE$COPA_PAT (1 REFER(LE$COPA_DIM)),
        3 BE$COPA_PATT BIT(32) ALIGNED,     /* test pattern          */
        3 BE$COPA_INV  BIT(32) ALIGNED;     /* invert pattern        */
/* -------------- end of SE$COPA --------------------------------- */
```

Similar to the windows, the pattern conditions may keep n subpatterns. Up to eight may be checked in a Dynamic List and up to four in macro $COND. Each subpattern is compared to a different object which can be any Data Element Member of Type BIT(16) or BIT(32) ALIGNED. The condition is TRUE if all subpatterns match. There are four matching modes:

1. IDENT
   Pattern and object must be identical.
2. ANY
   Pattern and object must have at least one common bit set.
3. INCL
   TRUE if all bits set in the pattern are set in the object (like IDENT inclusive additional bits set only in the object).
4. EXCL
   TRUE if all bits set in the object are set in the pattern (like ANY exclusive additional bits set only in the object).

In addition single bits in the objects can be inverted before testing.

Create Pattern Conditions

```
DBM> CRE COND PATTERN name pattern dimension INVERT=pattern
DBM> CRE COND PATTERN c '1'B               ! 1 subpattern
                                            ! padded right with 0
DBM> CRE COND PATTERN c '1'B 2             ! 2 subpatterns 32 bit each
DBM> CRE COND PATTERN c '1'B INV='1'B /ANY ! invert first bit
DBM> CRE COND PATTERN c '11111'B /IDENT    ! identical match
```

Set Pattern Condition Patterns

```
DBM> SET COND PATTERN name pattern invpat index
DBM> SET COND PATTERN c '1'B INDEX=2        ! 2. subpattern
DBM> SET COND PATTERN c '1'B '1'B           ! invert first bit
```

10.1.6 Function Conditions

Data Elements

Function conditions are implemented in two Data Elements. The declarations are kept in library GOOTYP(SE$COHE) and GOOTYP(SE$COFU).

```
/* ----------- start of SE$COFU -------------------------------------*/
DCL 1 SE$COFU BASED, /* function condition counters and attributes    */
        2 LE$COFU_TRUE_CT BIN FIXED(31) , /* true count */
        2 LE$COFU_FALSE_CT BIN FIXED(31) , /* test count */
        2 CVE$COFU_IMAGE CHAR(14) VAR,  /* sharable image logical name*/
        2 CVE$COFU_MODULE CHAR(30) VAR, /* Module name                */
        2 LE$COFU_DIM  BIN FIXED(31),   /* number of arguments        */
        2 SE$COFU_ARGS (1 REFER(LE$COFU_DIM)),
          3 CVE$COFU_BASE CHAR(14) VAR,    /* Base of argument */
          3  LE$COFU_DIR_ID BIN FIXED(31), /* Directory index  */
          3  LE$COFU_ARG_ID BIN FIXED(31), /* index            */
          3  LE$COFU_ARG_VER BIN FIXED(31);/* version          */
/* -------------- end of SE$COFU ---------------------------- */
```

The user may write his own routines for more complex conditions. These routines must be linked in a sharable image (DCL command `LSHARIM`) and can then be dynamically loaded. In the Dynamic List any members of Data Elements may be specified as arguments for these routines. The first argument, however, must be a BIT(8) ALIGNED returning the result.

Create Function Conditions

```
   DBM> CRE COND FUNC c
```

NOTE: presently the image, module and the arguments can be specified only in the Dynamic List Entry, but not in the condition.

10.1.7 Polygon Conditions

Data Elements

Polygon conditions are implemented in two Data Elements. The declarations are kept in library GOOTYP(SE$COHE) and GOOTYP(SE$COPO).

```
/* ------------ start of SE$COPO --------------------------------- */
DECLARE
 1 SE$COPO BASED,                   /* polygon condition attribute  */
   2 LE$COPO_TRUE_CT BIN FIXED(31) , /* true count                  */
   2 LE$COPO_FALSE_CT BIN FIXED(31) ,/* test count                  */
   2 CVE$COPO_BASE CHAR(14) VAR,     /* base of polygon               */
   2 RE$COPO_FACTOR BIN FLOAT(24),   /* factor for objects            */
   2 RE$COPO_OFFSET BIN FLOAT(24),   /* offset for objects            */
   2 RE$COPO_BINSIZE BIN FLOAT(24),  /* binsize for table             */
   2 LE$COPO_DIR_ID BIN FIXED(31),   /* polygon Directory index       */
   2 LE$COPO_POLH_ID BIN FIXED(31),  /* polygon header index          */
   2 LE$COPO_POLD_ID BIN FIXED(31);  /* polygon data index            */
/* ------------- end of  SE$COPO  --------------------------------- */
```

A polygon is created and modified independent of polygon conditions. Therefore several polygon conditions may reference the same polygon, but with different objects (coordinates). The polygon and objects are bound to the condition either by creation or by inserting in the Dynamic List or by macro call. The execution time is similar to window conditions.

Create Polygon Conditions

```
   DBM> CRE COND POLY c polygon       ! polygon must exist
```

The polygon referenced by a condition can be modified by command `REPLACE POLYGON` which is executed in the display program.

10.1.8 Composed Conditions

Data Elements

Composed conditions are implemented in three Data Elements. The declarations are kept in library GOOTYP(SE$COHE), GOOTYP(SE$COCO) and GOOTYP(SE$COCOT).

```
/* ------------ start of  SE$COCO ---------------------------------- */
DECLARE
 1 SE$COCO BASED,                     /* composed condition           */
   2 LE$COCO_TRUE_CT BIN FIXED(31) ,  /* true count                   */
   2 LE$COCO_FALSE_CT BIN FIXED(31) , /* test count                   */
   2 CVE$COCO_SPEC CHAR(62) VAR,      /* Condition spec.string        */
   2 LE$COCO_DIM BIN FIXED(31),       /* number of conditions         */
   2 SE$COCO_LIST (1 REFER(LE$COCO_DIM)),
     3 CVE$COCO_BASE CHAR(14) VAR,    /* base of condition            */
     3  LE$COCO_DIR_ID BIN FIXED(31), /* Directory index              */
     3  LE$COCO_CON_ID BIN FIXED(31), /* condition index              */
     3  LE$COCO_CON_VER BIN FIXED(31); /* condition version           */
/* ------------- end of  SE$COCO ---------------------------------- */
/* ------------ start of SE$COCOT --------------------------------- */
DECLARE
 1 SE$COCOT BASED,                   /* result table                 */
```

```
   2 LE$COCOT_TABSIZE BIN FIXED(31),  /*  2**(LE$COCO_NO)-1          */
   2 BE$COCOT_TABLE (0:1 REFER(LE$COCOT_TABSIZE)) BIT(1) ALIGNED;
                                    /* look-up table for complex cond.
                                         has 2**(LE$COCO_DIM) entries */
/* -------------- end of  SE$COCOT ---------------------------------- */
```

This may be any boolean expression of other conditions.

Create Composed Conditions

```
   DBM> CRE COND COMP c "a | (x & y)"  ! conditions a, x, y must exist
```

10.2 Polygons

10.2.1 Related Commands

```
CREATE  POLYGON
DELETE  POLYGON
SHOW    POLYGON
COPY    POLYGON
REPLACE POLYGON
DISPLAY POLYGON
```

10.2.2 Data Elements

Polygons are implemented in two Data Elements. The declarations are kept in library GOOTYP(SE$POHE) and GOOTYP(SE$PODAT).

```
/* --------------- begin of SE$POHE  ------------------------ */
DCL 1 SE$POHE BASED,
    2 LE$POHE_VERSION BIN FIXED(31), /* version number       */
    2 LE$POHE_DATA_ID BIN FIXED(31), /* index of SE$PODAT     */
    2 LE$POHE_POINTS  BIN FIXED(31), /* number of points      */
    2 LE$POHE_USED    BIN FIXED(31), /* number of used points */
    2 BE$POHE_FLAGS   BIT(32) ALIGNED; /* flags               */
/* ---------------- end of  SE$POHE ------------------------- */
/* --------------- begin of SE$PODAT ------------------------ */
DCL L_SE$PODAT_points BIN FIXED(31);
DCL 1 SE$PODAT BASED,
    2 RE$PODAT_xmin  BIN FLOAT(24), /* range                 */
    2 RE$PODAT_xmax  BIN FLOAT(24), /* range                 */
    2 RE$PODAT_ymin  BIN FLOAT(24), /* range                 */
    2 RE$PODAT_ymax  BIN FLOAT(24), /* range                 */
    2 LE$PODAT_points BIN FIXED(31),/* number of points      */
    2 LE$PODAT_used  BIN FIXED(31), /* number of used points  */
    2 RE$PODAT_x(L_SE$PODAT_points   REFER(LE$PODAT_points)) BIN FLOAT(24),
    2 RE$PODAT_y(L_SE$PODAT_points   REFER(LE$PODAT_points)) BIN FLOAT(24),
    2 HE$PODAT_key(L_SE$PODAT_points REFER(LE$PODAT_points)) BIN FIXED(7);
/* --------------- end of SE$PODAT -------------------------- */
```

Polygons may be created, displayed, modified, copied and deleted. They can be specified by graphic input or numerically. They are used by one or more conditions.

10.2.3 Create Polygons

```
DBM> CREATE POLYGON polygon points
```

The parameter 'points' defines the initial size of the polygon. If the polygon is modified, this size is extended automatically if needed.

10.2.4 Modify Polygons

Polygons can be modified only in the display program. If a two-dimensional spectrum or a scatterplot is displayed, the polygon can be edited by cursor input.

```
DISP> REPLACE POLYGON polygon frame
```

where 'frame' must be specified, if there are more than one frame on the screen. The points of the polygon can be entered by values, too.

```
DISP> REPLACE POLYGON polygon X=(x1,x2,x3,...) Y=(y1,y2,y3,...)
```

10.3 Spectra

10.3.1 Related Commands

CREATE	SPECTRUM
DELETE	SPECTRUM
SHOW	SPECTRUM
COPY	SPECTRUM
SUMUP	SPECTRUM
DISPLAY	SPECTRUM
FREEZE	SPECTRUM
UNFREEZE	SPECTRUM
CALCULATE	SPECTRUM
[DE]CALIBRATE	SPECTRUM
CLEAR	SPECTRUM
DUMP	SPECTRUM
FIT	SPECTRUM

10.3.2 Data Elements

Spectra are implemented in four Data Elements. The declarations are kept in library GOOTYP(SE$SPHE), GOOTYP(SE$SPHED), GOOTYP(SE$SPDTT), and GOOTYP(SE$SPDTD) where t is I,L or R and d is 1 or 2.

```
/* -------------------- start of SE$SPHE ------------------------ */
DCL 1 SE$SPHE BASED,                    /* spectrum header       */
      2 BE$SPHE_ATTR BIT(32) ALIGNED,   /* attribute flags       */
      2 CVE$SPHE_NAME CHAR(30) VAR,     /* name                  */
      2 LE$SPHE_DIM BIN FIXED(31) ,     /* no of dimensions      */
      2 CE$SPHE_DTYPE CHAR(1),          /* Data Type of spectrum */
      2 CVE$SPHE_DATE_TIME_CRE CHAR(23) VAR, /* creation date    */
      2 CVE$SPHE_CREATOR CHAR(62) VAR,
      2 LE$SPHE_SPTAB_DIR_ID BIN FIXED(31), /* table Directory   */
      2 LE$SPHE_SPTAB_DE_ID BIN FIXED(31),  /* table index       */
      2 LE$SPHE_ID_IN_SPTAB BIN FIXED(31);  /* spectrum index    */
/* ---------------------- end of SE$SPHE ------------------------ */
/* -------------------- start of SE$SPHED ----------------------- */
DCL 1 SE$SPHED BASED,
      2 BE$SPHED_ATTR BIT(32) ALIGNED;/* attribute flags longword */
/* ---------------------- end of SE$SPHED ----------------------- */
/* -------------------- start of SE$SPDTT ----------------------- */
DCL 1 SE$SPDTT BASED,
      2 BE$SPDTT_FLAGS BIT(32) ALIGNED,     /* flag table        */
      2 CVE$SPDTT_LETTERING_0 CHAR(80) VAR, /* first lettering   */
      2 IE$SPDTT_DIM_DATA BIN FIXED(15) ,   /* dimension         */
      2 SE$SPDTT_DATA (1 REFER(IE$SPDTT_DIM_DATA)),
        3 CVE$SPDTT_LETTERING CHAR(80) VAR,     /* lettering     */
        3 RE$SPDTT_LIMITS_LOW BIN FLOAT(24),    /* lower limit   */
        3 RE$SPDTT_LIMITS_UP BIN FLOAT(24),     /* upper limit   */
        3 LE$SPDTT_NO_BINS BIN FIXED(31),       /* no of bins    */
        3 RE$SPDTT_LIMITS_BIN BIN FLOAT(24),    /* bin size      */
        3 RE$SPDTT_FACTOR  BIN FLOAT(24),       /* coord. to ch. */
        3 RE$SPDTT_OFFSET  BIN FLOAT(24);       /* coord. to ch. */
/* ------------------- end  of SE$SPDTT ------------------------ */
/* ------------------ start of SE$SPDL1     --------------------- */
/* example of data structure of                                  */
/*  one dimensional FOUR bytes integer spectrum                  */
/* ------------------------------------------------------------- */
DCL 1 SE$SPDL1 BASED, /* one dimensional scalar spectrum         */
      2 LE$SPDL1_COUNTS BIN FIXED(31),            /* sum of counts      */
      2 DE$SPDL1_CONTENTS BIN FLOAT(53),          /* contents           */
      2 LE$SPDL1_OUTLIM_UP_COUNTS BIN FIXED(31),  /* overflow counts    */
      2 LE$SPDL1_OUTLIM_LOW_COUNTS BIN FIXED(31), /* underflow counts   */
      2 DE$SPDL1_OUTLIM_UP_CONTENTS BIN FLOAT(53), /* overflow contents  */
      2 DE$SPDL1_OUTLIM_LOW_CONTENTS BIN FLOAT(53),/* underflow contents */
      2 LE$SPDL1_DIM1 BIN FIXED(31),              /* channels dim 1     */
      2 LE$SPDL1_DATA (1 REFER(LE$SPDL1_DIM1)) BIN FIXED(31);
/* -------------------- end of SE$SPDL1    ---------------------- */
```

By default, spectra are kept in the Directory $SPECTRUM. Each spectrum takes four entries. The user need not be concerned with that, but in a SHOW DIRECTORY command these Data Elements will be listed. Spectra may be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24). The dimensionality can be up to two. Spectra may be filled in a Dynamic List Entry or by macro $ACCU. Default Pool is $SPEC_POOL.

Spectra are created as digital, CAMAC or analog spectra.

- Digital spectra are used to accumulate integer or bit variables. The integer binsize specifies the number of input bins to be incremented in one spectrum bin. Bit spectra should be dimensioned (1,16) or (1,32), respectively, with binsize 1.

- CAMAC spectra are incremented in CAMAC memory (MR2000). The contents of the CAMAC memory must be copied into GOOSY spectra. These GOOSY spectra are created as digital spectra with binsize 1.

- Analog spectra are used to accumulate float variables. The binsize specifies an interval. The lower limit of the interval is inclusive, the upper limit exclusive. Therefore the upper spectrum limit is exclusive.

10.3.3 Create Spectra

```
DBM> CRE SPEC s L (0,1023) 10 /DIGITAL      ! BIN FIXED(31), binsize=10
DBM> CRE SPEC s R (0,1023,0,255) (10,10)    ! BIN FLOAT(24), 2-dim.
DBM> CRE SPEC s(10) L (-10,15) 0.1 /ANALOG  ! name array, binsize 0.1
```

10.3.4 CAMAC Spectra

CAMAC spectra are normal digital GOOSY spectra. They keep additional information about the location of the CAMAC spectrum data in the CAMAC crate. This information must be specified during creation of the spectrum.

Related Commands

Some additional commands are provided to access CAMAC spectra, i.e. to copy the data from/to CAMAC and to clear the CAMAC data. The accumulation of the CAMAC spectra is controlled by special `START` and `STOP` commands. Note that these commands are executed in the Data Base Manager which must therefore be running.

```
DBM> CRE SPEC c L (0,1023) BRANCH=0 CRATE=1 STATION=20 OFFSET=1024/CAMAC
DBM> START MR2000 branch crate station /INIT
DBM> STOP  MR2000 branch crate station
DBM> CLEAR CAMAC SPECTRUM name /CAMAC/SPECTRUM
DBM> READ  CAMAC SPECTRUM name /ADD
DBM> WRITE CAMAC SPECTRUM name /ADD
DBM> SHOW  CAMAC SPECTRUM name      !contents of MR2000
```

CAMAC Spectrum Data Element

The additional information is stored in an additional Data Element GOOTYP(SA$SPCAM).

```
/* -------------------- begin of SE$SPCAM     ---------------- */
DCL 1 SE$SPCAM BASED,
    2 BE$SPCAM_mask    BIT(16) ALIGNED, /* bit mask           */
    2 IE$SPCAM_branch  BIN FIXED(15),   /* branch             */
    2 IE$SPCAM_crate   BIN FIXED(15),   /* crate              */
    2 IE$SPCAM_station BIN FIXED(15),   /* station            */
    2 LE$SPCAM_start   BIN FIXED(31),   /* offset in bytes    */
    2 LE$SPCAM_length  BIN FIXED(31);   /* length in longwords */
/* -------------------- end of   SE$SPCAM     ---------------- */
```

The spectra are referenced as normal spectra.

10.3.5 Spectrum Calibration

See also section 10.4.

Related Commands

A calibration is connected/disconnected to a spectrum by

```
    DBM> CALIBRATE SPECTRUM spectrum calibration
    DBM> DECALIBRATE SPECTRUM spectrum
```

Spectrum Calibration Data Element

One more Data Element is queued to the spectrum header:

```
/*--------------- start of SE$SPCAL  -----------------------*/
DCL P_SE$SPCAL POINTER;
DCL 1 SE$SPCAL BASED (P_SE$SPCAL),
      2 L_SE$SPCAL_DIM BIN FIXED(31),
      2 SE$SPCAL_DIM(1 REFER (L_SE$SPCAL_DIM)),
        3 L_SE$SPCAL_DIR_ID BIN FIXED(31),
        3 L_SE$SPCAL_POOL_ID BIN FIXED(31),
        3 L_SE$SPCAL_DE_ID BIN FIXED(31);
/*--------------- end of    SE$SPCAL ----------------------*/
```

10.3.6 Dump Spectra for IBM

GOOSY spectra can be dumped to a VAX disk file and then copied into an IBM VSAM library to be read later into SATAN programs.

```
     DBM> DUMP SPECTRUM name OUTPUT=file
```

Several spectra specified by wildcards can be dumped and transferred at once. The spectrum files are transferred to the IBM by the DCL-command

```
     $ SIBMSPEC file VSAMlib
```

The VSAM library must exist on the IBM!

10.4 Calibrations

10.4.1 Related Commands

```
CREATE  CALIBRATION type
SHOW    CALIBRATION
SET     CALIBRATION type (display program)
DISPLAY CALIBRATION (display program)
```

10.4.2 Calibration Data Elements

By default, calibrations are kept in the Directory $CALIB. Similar to spectra calibrations are sets of several Data Elements. They keep a calibration table which is used to calibrate the spectra data when displaying them. Each calibration can be connected to an arbitrary number of spectra.

```
/* -------------------- begin of SE$CAHE     ---------------- */
DCL P_SE$CAHE   POINTER;
DCL 1 SE$CAHE BASED(P_SE$CAHE),
      2 L_SE$CAHE_POINTS BIN FIXED(31),
      2 L_SE$CAHE_USED BIN FIXED(31),
      2 L_SE$CAHE_CAL_ID BIN FIXED(31),
      2 B_SE$CAHE_MASK BIT(32) ALIGNED;
/* -------------------- end of   SE$CAHE     ---------------- */
/* -------------------- begin of SE$CADA     ---------------- */
DCL P_SE$CADA POINTER;
DCL 1 SE$CADA BASED(P_SE$CADA),
      2 B_SE$CADA_MASK  BIT(32) ALIGNED,
      2 CV_SE$CADA_UNITS CHAR(78)VAR,
      2 L_SE$CADA_USED  BIN FIXED(31),
      2 L_SE$CADA_UNCAL  BIN FIXED(31),
      2 L_SE$CADA_CAL  BIN FIXED(31),
      2 R_SE$CADA_UNCAL (1 REFER(L_SE$CADA_UNCAL))
                         BIN FLOAT(24),
      2 R_SE$CADA_CAL (1 REFER(L_SE$CADA_CAL))
                         BIN FLOAT(24);
/* -------------------- end of   SE$CADA     ---------------- */
```

Each calibration takes two entries in Directory $CALIB. One for the main Data Element and one entry for the calibration table contents. First a calibration is created. Then it is connected to several spectra.

10.4.3 Fixed Calibrations

The calibration table contains calibrated values with a fixed step width for the uncalibrated values. The calibrated values are specified by `SET CALIBRATION FIXED` command. They can be specified by values, parameters for a polynomial, read from a file or calculated by a user written procedure.

Create Fixed Calibrations

```
DBM> CREATE CALIBRATION FIXED name entries
```

The parameter 'entries' specifies the number of calibrated values in the table.

Set Fixed Calibration Parameters

```
DBM> SET CALIBRATION FIXED name
```

A more detailed description of this command is found in the GOOSY Display Manual.

10.4.4 Float Calibrations

The calibration table contains uncalibrated and calibrated values. These values are specified by the `SET CALIBRATION FLOAT` command. They can be read from a file or calculated by a user written procedure.

Create Float Calibrations

```
DBM> CREATE CALIBRATION FLOAT name entries
```

The parameter 'entries' specifies the number of calibrated and uncalibrated pairs in the table.

Set Float Calibration Parameters

```
DBM> SET CALIBRATION FLOAT name
```

A more detailed description of this command is found in the GOOSY Display Manual.

10.4.5 Linear Calibrations

Linear calibrations do not use a table, but two parameters for a linear polynomial. A table is not needed because linear polynomials can be inverted. The calibration parameters are set by command `SET CALIBRATION LINEAR`.

Create Linear Calibrations

```
DBM> CREATE CALIBRATION LINEAR name
```

Set Linear Calibration Parameters

```
DBM> SET CALIBRATION LINEAR name
```

A more detailed description of this command is found in the GOOSY Display Manual.

10.5 Pictures

10.5.1 Related Commands

```
CREATE PICTURE
DELETE PICTURE
SHOW   PICTURE
CLEAR  PICTURE
MODIFY FRAME
```

10.5.2 Data Elements

By default, pictures are kept in the Directory $PICTURE. They keep information to display several frames containing spectra or scatterplots. Up to 64 frames may be displayed on one screen. Pictures take one entry in the $PICTURE Directory. Each frame takes one more entry. The default Pool is $PIC_POOL.

10.5.3 Create Pictures

When a picture is created, only the number of frames is specified. The frame content must then be specified by subsequent commands `MODIFY FRAME`. The same frame can be modified several times.

```
DBM> CRE PICTURE name frames
DBM> CRE PICTURE pict 6 /NOPROMPT          ! 6 frames
```

Without the `/NOPROMPT` qualifier the command will enter a menu driven prompting loop to specify all frames. Therefore this qualifier must be specified in command procedures!

10.5.4 Modify Picture Frames

Two commands are provided to modify frames, one for scatter frames and one for spectrum frames:

```
DBM> MOD FRAME SCATTER picture frame x y limits
DBM> MOD FRAME SPECTRUM picture frame spectrum limits
DBM> MOD FRAME SCATTER pict 1 [DATA]evt.geli(1) [DATA]evt.naj(1)
                                       ! frame one scatter
DBM> MOD FRAME SPECTRUM pict 3 [$SPECTRUM]s
                                       ! frame three spectrum
```

A more detailed description of this command is found in the GOOSY Display Manual.

10.6 User Data Elements

10.6.1 Related Commands

```
CREATE ELEMENT
DELETE ELEMENT
SHOW   ELEMENT
COPY   ELEMENT
CLEAR  ELEMENT
```

10.6.2 Create Data Elements

Besides the GOOSY Data Elements the user may define and create his own Data Elements. This may be done by GOOSY commands or by subroutine calls in a program. The following steps must be performed:

1. Put the PL/I source declaration of the Data Element in a text library. The name of the structure should be used as the name for the library module. The declaration must declare a based structure. A base pointer may be specified.
2. Create a Directory in Data Base
3. Create a Pool in Data Base
4. Create the Data Element Type, using the new PL/I structure in the text library.
5. Create the Data Element of the new Type.

If the declaration contains REFER members, the Data Element can be created only in a program, because the REFER values must be specified. To access the Data Element in an analysis procedure, include the library module declaring its structure and call $LOC macro to receive the pointer to the Data Element. In stand alone private programs the macro $ATTACH must be called to attach the Data Base first.

10.6.3 Example

Assume we want to create a Data Element like this:

```
DCL P_SX$evt POINTER;
DCL 1 SX$evt BASED(P_SX$evt),
    2 patt     BIT(32) ALIGNED,
    2 geli(10) BIN FIXED(31),
    2 naj(10   BIN FIXED(15);
```

The structure must be BASED in order to use it in a PL/1 program as well. This declaration is in our library TPRIV in module SX$EVT. We create in the following example a Data Element EVT of the above Type.

```
DBM> CRE TYPE @tpriv(SX$evt)                  ! Declaration in library
DBM> CRE ELEMENT [DATA]evt evtdata SX$evt      ! Directory DATA and
                                              !  Pool EVTDATA must exist
```

10.7 Dynamic Lists

10.7.1 Related Commands

```
CREATE DYNAMIC LIST
DELETE DYNAMIC LIST
SHOW   DYNAMIC LIST
CREATE DYNAMIC ENTRY type
DELETE DYNAMIC ENTRY type
```

10.7.2 Create Dynamic Lists

Each Dynamic List takes two entries in Directory $DYNAMIC. The default Pool is $DYNAMIC.

```
DBM> CRE DYNAMIC LIST list 100     ! Dynamic List for 100 entries
```

The following switches apply for the `CREATE DYNAMIC ENTRY` commands:

`/UPDATE` The modification becomes active immediately (also for the `DELETE DYNAMIC ENTRY` command) for a running analysis.

`/MASTER` Valid for conditions (except multiwindow) and procedures. Master Functions are executed first of all other Entries. Master conditions are executed first of all other conditions. If a master conditions result is false, the Dynamic List execution is terminated. If the same master condition is in two Dynamic Lists, both lists are skipped, if the condition was false.

In all commands explicit defaults for Data Base, node and Directories can be specified. These parameters are not included in the following descriptions:

```
DYN_DIR=default Directory of Dynamic List
COND_DIR=default Directory of condition
SPEC_DIR=default Directory of spectrum
PAR_DIR=default Directory of parameters
POLY_DIR=default Directory of polygon
BASE=default Data Base
NODE=default node
```

10.7.3 Arrays

Spectra or conditions may be arrays. In this case an index range must be specified. All additional Data Elements must be either scalar or indexed by the same range. Ranges are specified by (lower limit : upper limit).
Examples:

```
DBM> CRE DYN ENTRY WINDOW dlist [d]e_recoil(1:5) -
          PARA=[d]$event.ener
DBM> CRE DYN ENTRY SPECTRUM dlist [d]ener1(2:4) -
           PARA=[d]$event.e(2:4) CONDI=[d]de_window
DBM> CRE DYN ENTRY SPECTRUM dlist [d]ede(1:4) -
           PARA=([d]$event.e,$event.de)
DBM> CRE DYN ENTRY INDEXED dlist [d]ede(1:7) -
           PARA=([d]$event.e,$event.de) -
           INDEX=[d]a.b(1)
```

[d] is the Directory specification
The difference between windows and multiwindows is that multiwindows have only one object for all subwindows, but one result bit for each, whereas windows need one object per subwindow, but have only one result bit (set, if all subwindows are true). Multiwindows may be used as filters for spectrum array accumulation. The internal dimension of the window must match the specified index range. It may also be used for 'indexed' spectrum accumulation. Then the index of the last matching subwindow is used to select the spectrum member. In the first case, the subwindows may overlap, in the second case this makes normally no sense.

```
DBM> CRE DYN ENTRY SPECTRUM list [d]ener1(2:4) -
           PARA=[d]$event.e(2:4) CONDI=[d]m_window
! three spectrum Entries are executed
DBM> CRE DYN ENTRY INDEXEDSPECTRUM list [d]ener(2:4) -
           PARA=[d]$event.e(1) INDEX=[d]m_window
! One spectrum Entry is executed
```

[d] is the Directory specification
In both cases 'm_window' must have 3 subwindows.

10.7.4 Creating Dynamic List Entries

PROCEDURE

Command to insert an entry with a user specified procedure call:

```
CRE DYN ENTRY PROCEDURE listname MODULE=image(module)
                                 PARAMETER=(argument list)
                                 CONDITION=cond
                                 /MASTER
MODULE        module specification as 'image(module)'. Module
              must be linked in sharable image
image         logical name of shar.image
PARAMETER     arg.list of DE-members
CONDITION     name of condition (optional)
/MASTER       master Entry
```

This Entry will call a module from a sharable image. The pointers to the Data Elements specified in the argument list are passed to the procedure.
Example:

```
CRE DYN ENTRY PROCEDURE dlist
             MOD=privshar(x$loop)
             PARA=([d]$event.z4.de(5),$event.z5)
             /MASTER
```

[d] is the Directory specification
The X$LOOP declaration must be:
`ENTRY(POINTER,POINTER) RETURNS(BIN FIXED(31))`
The sharable image must be linked by the DCL command `LSHARIM`.

FUNCTION

Command to insert an entry with a user specified condition function call:

```
CRE DYN ENTRY FUNCTION listname condition MODULE=image(module)
                                PARAMETER=(argument list)
                                /MASTER
MODULE        module specification as 'image(module)'. Module
              must be linked in sharable image. Is used and required only
              if not specified in condition.
image         logical name of shar.image
PARAMETER     arg.list of DE-members
/MASTER       master entry
```

This entry will call a module from a sharable image. The pointers to the Data Elements specified in argument list are passed to the procedure. The first argument, a BIT(1) ALIGNED, returns the condition result.
Example:

```
CRE DYN ENTRY FUNCTION dlist [d]cond
              MOD=privshar(x$cond)
              PARA=([d]$event.z4.de(5),$event.z5)
```

[d] is the Directory specification
The X$COND declaration must be:
`ENTRY(BIT(1) ALIGNED,POINTER,POINTER) RETURNS(BIN FIXED(31))`
The sharable image must be linked by the DCL command `LSHARIM`.

PATTERN

Command to insert a pattern condition entry:

```
CRE DYN ENTRY PATTERN listname cond PARAMETER=object
                                    /MASTER
PARAMETER     DE-members
/MASTER       Master entry
```

The entry will check a specified Data Element member versus a pattern. Note that four test modes can be specified with the pattern condition (IDENT, ANY, EXCL, INCL). The values of the Data Element members can be inverted bitwise. Up to 8 internal dimensions. Objects can be of type BIT(16), BIT(32), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY PATTERN dlist [d]main_pat
         PARA=[d]$event.pat
         /MASTER
```

[d] is the Directory specification

WINDOW

Command to insert a window condition entry:

```
CRE DYN ENTRY WINDOW listname cond PARAMETER=object
                                   /MASTER
PARAMETER     DE-members
/MASTER       Master entry
```

This entry will check a specified Data Element member versus window limits. Up to 8 internal dimensions. The objects may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY WINDOW dlist [d]e_recoil
         PARA=[d]$event.ener
```

[d] is the Directory specification

MULTIWINDOW

Command to insert a multiwindow condition entry:

```
CRE DYN ENTRY MULTIWINDOW listname cond PARAMETER=object

PARAMETER     DE-member
```

This entry will check a specified Data Element member versus all window limits. For each check a result bit is set, which may be used to increment a spectrum array member. In addition, the number of the last matching window may be used as the index of a spectrum array member (see INDEXEDSPECTRUM). The object may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY MULTI dlist [d]e_recoil
         PARA=[d]$event.ener
```

[d] is the Directory specification

POLYGON

Command to insert a polygon condition entry:

```
CRE DYN ENTRY POLYGON listname cond PARAMETER=(x,y)
                                     POLYGON=name
                                     /MASTER
PARAMETER     DE-members for X and Y. Used and required only
              if not specified in condition.
POLYGON       Name of polygon. Used and required only
              if not specified in condition.
/MASTER       Master entry
```

It is checked whether the point X,Y is inside (true) or outside (false) the polygon. Objects may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY POLY dlist [d]poly_1
         PARA=([d]$event.de,[d]$event.ener)
         POLYG=poly
```

[d] is the Directory specification.

COMPOSED

Command to insert a composed condition entry:

```
CRE DYN ENTRY COMPOSED listname cond /MASTER

/MASTER       Master entry
```

A boolean expression of conditions is executed. The expression is specified in the corresponding condition Data Element.
Example:

```
   CRE DYN ENTRY COMPOSED dlist [d]all_ok /MASTER
```

[d] is the Directory specification

SPECTRUM

Command to insert a spectrum entry:

```
   CRE DYN ENTRY SPECTRUM listname spectrum PARAMETER=object
                                            CONDITION=cond
                                            INCREMENT=incr
   PARAMETER     DE-members for coordinates
   CONDITION     condition for spectrum (optional)
   INCREMENT     DE-member for increment (optional) (BIN FLOAT(24))
```

Supports spectra of Type BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24) with up to 2 dimensions. Coordinates can be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24).
Examples:

```
   CRE DYN EN SPECTRUM dlist [d]ener1
              PARA=[d]$event.e(1) CONDI=[d]de_window
   CRE DYN EN SPECTRUM dlist [d]ede
              PARA=([d]$event.e,$event.de)
```

[d] is the Directory specification

INDEXEDSPECTRUM

Command to insert an indexed spectrum entry:

```
   CRE DYN ENTRY INDEXEDSPECTRUM listname spectrum(l:u) PARAMETER=object
                                                        INDEX=index
                                                        INCREMENT=incr
                                                        CONDITION=cond
   PARAMETER     DE-members for coordinates
   INDEX         DE-member (BIN FIXED(31)) or multiwindow to specify
                 spectrum member
   CONDITION     condition for spectrum (optional)
   INCREMENT     DE-member for increment (optional) (BIN FLOAT(24))
```

Supports spectra of Type BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24) with up to 2 dimensions. Coordinates can be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24). Specified spectrum must be an array. Specification of index is used to select the spectrum member to be incremented. This could be either a parameter Data Element or a multiwindow.
Examples:

```
   CRE DYN EN INDEXED dlist [d]ener(1:10)
              PARA=[d]$event.e(1) INDEX=[d]m_window
   CRE DYN EN INDEXED dlist [d]ede(1:5)
              PARA=([d]$event.e,$event.de) INDEX=[d]a.b
```

[d] is the Directory specification

BITSPECTRUM

Command to insert a bitspectrum entry:

```
   CRE DYN ENTRY BITSPECTRUM listname spectrum PARAMETER=object
                                                CONDITION=cond
   PARAMETER     DE-members for coordinates
   CONDITION     condition for spectrum (optional)
```

Supports one dimensional spectra, Type BIN FIXED(31). Coordinates must be BIT(32), BIT(16), BIN FIXED(15), or BIN FIXED(31).
Example:

```
   CRE DYN ENTRY BIT dlist [d]patt
             PARA=[d]$event.pat(1)
```

[d] is the Directory specification

SCATTER

This entry is inserted by the `DISPLAY PICTURE` command, if scatter frames are in the picture, or by the `DISPLAY SCATTER` command. The name of the list can be specified optionally with these commands. The default is $SCATTER. Note that this list must be attached to be active. It should be attached as the last list. Scatter Entries are deleted only by the creating display process. This may lead to 'dead' scatter Entries, i.e. if the environment name is no longer used. Attaching the list in this case a message is displayed that a link could not be opened. Then one should delete all scatter Entries of all types by the command:

```
   DELETE DYNAMIC ENTRY SCATTER list * * /UPDATE
```

No scatter plot should be active during that command.

10.7.5 Delete Entries

The commands to delete Dynamic Entries are similar to the create commands.

```
   DBM> DELETE DYNAMIC ENTRY type list entry
```

where 'type' is the same keyword as the fourth key in the create commands, 'list' is the Dynamic List name and 'entry' the name of the Data Element, e.g. the spectrum or condition. Different to the create commands the 'type' and 'entry' parameter may be wildcarded by an asterisk to delete all Entries or all Entries of a specific Type.

10.7.6 Examples

For Dynamic List Entries the objects for spectrum accumulation and condition checks, the spectrum increment and the index must be Members of GOOSY Data Elements created already in the Data Base. Assume we have created a Data Element like this:

```
DCL 1 SX$evt,
    2 patt     BIT(32) ALIGNED,
    2 geli(10) BIN FIXED(31),
    2 naj(10   BIN FIXED(15);
```

This declaration is in our library TPRIV in module SX$EVT. We refer in the following examples to Data Element EVT of the above Type. We assume that conditions c,w,a(1:10) and spectra s and s2 already exist.

```
DBM> CRE TYPE @tpriv(SX$evt)                ! Declaration in library
DBM> CRE ELEMENT [DATA]evt evtdata SX$evt    ! Directory DATA and
                                            !  Pool EVTDATA must exist

DBM> CRE DYNAMIC LIST list ENTRIES=100     ! Dynamic List for 100 Entries
DBM> CRE DYN ENTRY PATTERN list c PARAMETER=evt.patt
DBM> CRE DYN ENTRY WINDOW  list w PARAMETER=evt.geli(3)
DBM> CRE DYN ENTRY WINDOW  list a(1:10) PARA=evt.geli(1:10)
                                       ! condition name array!
DBM> CRE DYN ENTRY SPECTRUM list s PARAMETER=[DATA]evt.naj(1)
DBM> CRE DYN ENTRY SPECTRUM list s2 -
              PARAMETER=([DATA]evt.naj(1),[DATA]evt.geli(1)) -
              CONDITION=c
                                       ! 2-dim. spectrum
```

Part IV

GOOSY Display

Chapter 11

GOOSY Display Introduction

This manual presents to you an overview about the main facilities offered by the GOOSY Display. The description of the whole functionality would overcharge the extent of this manual. Therefore the concept and the main functions of the GOOSY display are shortly explained. A short description of all display commands is given in Appendix A on page **??**. For detail information about the GOOSY display commands you are referred to the **GOOSY Display Command** manual.

The GOOSY Display is activated by :

```
  $ MDISP                 ! in stand alone mode
or as a component in an GOOSY environment by
  $ GOOSY CREATE PROCESS DISP $DSP
```

The process is started by default with priority 3. Specify another priority by `CREATE PROCESS ... PRIO=p`. To summarize the possibilities of the GOOSY Display, a short description of the fundamental ideas is given:

Devices Are graphical terminals which are used for all graphical outputs. In the GOOSY Display up to 10 different graphical output devices can be operated in parallel.

Spectra and Scatterplots Single spectra or scatterplots can be displayed in different modes. The definition of global display representations is possible.

Pictures Several spectra and scatterplots can be displayed together, if they are organized in pictures. Pictures are elements in your Data Base (see the **GOOSY Data Management** manual), therefore the picture definitions are available even if the display process terminates. Each spectrum or scatterplot is imbedded in frames, which can be modified separately.

Frames The basic entities of a picture. It is possible to manipulate single frames without distoring the other components on the screen.

Calibrations Each calibration can be connected to one or several spectra. Displaying the corresponding spectra, the calibration is applied in the appropriate modes, if it is requested.

Overlays Beside the overlay of a single spectrum, it is possible to define any number of overlays for each frame in a picture. These overlays are displayed by a single command.

Projection The projection of two dimensional spectra on any axis, with arbitrary projection windows and projection polygons is implemented.

Updates The continuous update of any frame on the screen in a specified delta time is possible.

Display Versions Two different display versions are available. The *standard* implementation with the complete functionality, and the *fast* version, which consums only 60% of the CPU–time of the standard version, but with a reduced functionality (see section 18.

The Data Elements referred in the examples of this manual are GOOSY Data Element name specifications (see **GOOSY Data Management** manual). Directory and Data Base names are not explicitly indicated, except they have to be specified. In all commands missing Directory or Data Base names these are substituted by common default values; the standard GOOSY defaults or the user defined defaults. The standard GOOSY defaults are:

DB	for the Data Base name
$CALIB	for the calibration Directory
$CONDITION	for the condition Directory
$DYNAMIC	for the Dynamic List Directory
$PICTURE	for the picture Directory
$POLYGON	for the polygon Directory
$SPECTRUM	for the spectrum Directory
DATA	for the event Directory

All commands creating a Data Element need a Pool specification. The default Pools provided by the GOOSY-Display are:

$CAL_POOL	for the calibration Pool
$PIC_POOL	for the picture and polygon Pool
$SPEC_POOL	for the spectrum Pool

Chapter 12

Display Devices

The GOOSY Display offers the possibility to send the graphical output to up to 10 different devices simultaneously. At any time it is possible to connect or disconnect any device to or from the GOOSY Display. The graphical input occurs at one user definable master device.

12.1 Allocation of Devices

To generate graphical outputs the GOOSY Display has to be connected to a graphical output device by:

```
    ALLOCATE DEVICE name type
```

You have to specify the physical address "name" of the device and a GOOSY alias name "type" for the device type.

The GOOSY Display supports two different device classes. The first class contains the interactive devices (e.g. graphical terminals), at which each graphical output can be seen directly. The available device types are:

GOOSY Alias name	Device type
MG600	Monterey MG600 or MG620
	This is the default device type!
MV	Micro–VAX II graphic terminal
GPX	Micro–VAX II color graphic terminal
PECAD	Pecad terminal
TEK4014	Tektronix 4014
TEK4107,TEK4109,TEK4111,TEK4115	Tektronix 4107, 4109,4111,4115
VT240	VT 240, VT 241, VT 330, VT 340, REGIS terminals

Allocating such an interactive device the physical terminal address has to be known:

```
    ALLOCATE DEVICE txa9   pecad   ! terminal directly connected to the host
    ALLOCATE DEVICE lta999 tek4014 ! terminal connected to a DEC-server
```

To get the physical address of a terminal connected to a terminal server, login to the `Local>` terminal server mode (e.g. with the break key ⟨F5⟩), then type the server commands:

```
  Local> SHOW SERVER
  Local> SHOW PORT
  Local> LOGOUT
```

The server information shows you the device number range, e.g `LTA241-LTA248`. The port information gives you the port number, e.g `Port:  3`. The physical address of the graphical terminal yields to `LTA243`. Before creating any output at this terminal you have to logout from the server! Creating outputs at your current terminal specify as the physical address "SYS$COMMAND".

The second class contains the non–interactive devices (e.g. plotter). Allocating these devices, the specified physical address is interpreted as a file name, which is completed by a default file extension, if necessary. Available non–interactive devices are:

GOOSY Alias name	Device type	Default extension
LN03	LN03–Plus laser printer	.LN3
HP7550A4	HP7550A pen plotter with DINA4 sheets	.HP4
HP7550A3	HP7550A pen plotter with DINA3 sheets	.HP3
METAOUT	Output to metafile	.MET

To allocate a LN03 laser printer you have two possibilities:

```
  1.)  ALLOCATE DEVICE test ln03
  2.)  ALLOCATE DEVICE test.out ln03
```

The first case yields to the creation of the file "TEST.LN3" with device dependent sequences for the LN03–PLUS laser printer. The second example creates the file "TEST.OUT". Both can be plotted at any LN03–PLUS printer.

Beside the device dependent plotfiles you have the possibility to create a device independent metafile (e.g. "TEST.MET"):

```
    ALLOCATE DEVICE test.met metaout
```

It is possible to redisplay this metafile on any interactive or non–interactive device. Furthermore you have the possibility to send it to the IBM and to display it there at any available output device (see chapter 17.7 on page 241)!

If at least one device is allocated the GOOSY Display is ready for any graphical output.

12.2 The GOOSY Main Device

The GOOSY Display provides the possibility to allocate up to 10 graphical output devices; with the restriction that the device types are different. The graphical outputs are sent to all allocated devices simultaneously. But the requested graphical inputs have to be performed at one master device. This device is the GOOSY main device. By default the first allocated device is used as the GOOSY main device. If graphical inputs should occur at another device, allocate it with the "/MAIN" switch:

```
ALLOCATE DEVICE lta999 tek4014 /MAIN
```

In principle any device can be allocated as the GOOSY main device. But don't be surprised when no graphical input is possible using a plotter as the master device.

The main device is also used to adjust the graphical output. Normally the facilities of the allocated devices are different, e.g different character heights are supported. The GOOSY Display adjusts the graphical outputs using the character heights supported by the main device. Therefore in any case one main device has to be defined. The other device facilities, like the color, are supported in a device dependent manner.

12.3 Deallocation of Devices

If a plotfile or a metafile should be printed, it is necessary to close and to deallocate the file. In that case the corresponding device has to be disconnected from the GOOSY Display by:

```
DEALLOCATE DEVICE name
```

"name" is the device name as stored internally in the GOOSY Display. In the case of interactive devices it is the physical address, for non–interactive devices it is the file name; like "TEST.LN3". A list of all allocated devices and their internal GOOSY Display names can be get by:

```
SHOW DEVICES
```

The device names and types as well as the current main device type are listed. You also see which device is able to perform graphical inputs, e.g:

```
INFORMATION ON ALLOCATED DEVICES:
Device number:             1
  Logical device name : LTA999
  Physical device name: LTA999
  Device type         : Monterey MG600
  Device category     : Output and input

MAIN DEVICE TYPE        : Monterey MG600
```

Chapter 13

Spectra and Scatterplots

To display single spectra and scatterplots different commands have to be applied. Spectra can be displayed in many different modes, some are discussed in the following. The definition of global display modes for spectra is possible, they can be activated, deactivated, and modified at any time.
The display of scatterplots after a condition check against any parameter is possible. Scatterplots can run in different modes: asynchronously, or synchronously to the analysis.

13.1 Displaying Spectra

A single spectrum "name" could be displayed by:

```
DISPLAY SPECTRUM name
```

If nothing is specified the spectrum is displayed in the full spectrum range. Because each output device has a well defined resolution (e.g. 1024 pixels), it is not necessary to display more bins as pixels provided by the output device. Therefore the GOOSY Display optimizes the spectrum data by displaying them with an internal display binsize which is adjusted to the resolution of the GOOSY main device. Two binning modes are implemented:

- The average about several spectrum bins is calculated, the integral count rate remains unchanged. Therefore the displayed count rates are no longer integer values. Because this mode yields in a smooth spectrum contour it is activated by:

  ```
  DISPLAY SPECTRUM name /SMOOTH
  ```

- The maximum and minimum count rate of the gathered spectrum bins are displayed. With that mode spikes in the spectra are not lost. This mode is the default and can be activated by:

```
        DISPLAY SPECTRUM name /NOSMOOTH
```

The internal display binsize is activated if more than 1024 bins should be displayed.

Sometimes the automatic display binning is not desired. In that case you have the possibility to display a spectrum without binning by:

```
    DISPLAY SPECTRUM name BINFACTOR=1
```

If the binfactor is larger than 1 it specifies the number of spectrum bins to be summed up. This is identical to a temporary modification of the spectrum binsize. A binfactor of 0 yields to an automatic display binning. The actual binning is displayed in the spectrum information field on the screen.

A definite range of a spectrum is displayed specifying:

```
    DISPLAY SPECTRUM name 100,500        ! for one dimensional spectra
    DISPLAY SPECTRUM name 100,500,0,400  ! for two dimensional spectra
```

The scaling axis can be fixed by:

```
    DISPLAY SPECTRUM name SCALIM=100,700
```

Three different display modes (linear, logarithmic and squareroot) for each axis are available:

```
    DISPLAY SPECTRUM name /LOG   ! mode of scaling axis:
                                 ! y-axis for one dim. spectra
                                 ! z-axis for two dim spectra
    DISPLAY SPECTRUM name /XSQRT ! mode of x-axis
    DISPLAY SPECTRUM name /YLIN  ! mode of y-axis
```

All limits and qualifiers specified in one display command, the *temporary display modes*, are available until the next display and can be re–called by:

```
    DISPLAY SPECTRUM name /TEMP
```

The limits and qualifier specified in the display command have the highest priority and modify the corresponding values in the temporary display:

```
    DISPLAY SPECTRUM name /TEMP/LIN ! use temporary modes, but scaling axis
                                    ! is displayed linear.
```

For one dimensional spectra the following representations are implemented:

/HISTO The spectrum data are displayed as histograms.

/VECTOR The spectrum contents are connected by a polyline.

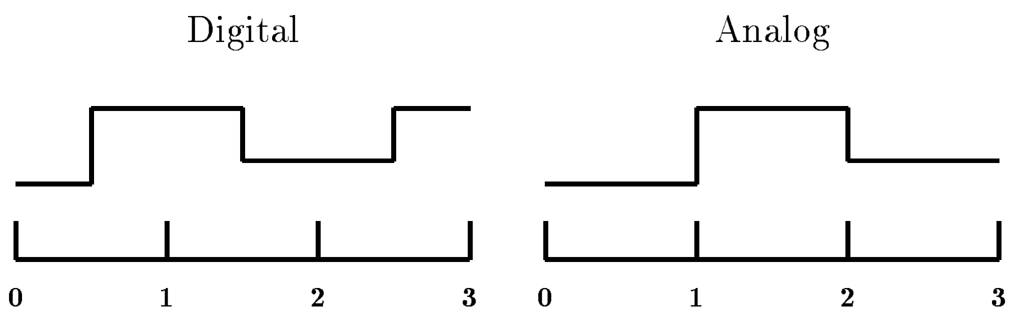


Figure 13.1: Display of Digital and Analogous Spectra

/MARKER The spectrum contents are indicated by markers.

For two dimensional spectra the following spectra representations are implemented:

/CLUSTER The spectrum count rates are indicated by clusters of different size and color.

/CONTOUR Contour lines are displayed.

/ISO A 3D plot is generated.

To display two dimensional spectra some additional parameters are necessary. For contour and cluster plots cuts or intervals have to be defined. They are specified in relative units of the spectrum maximum, e.g.

```
DISPLAY SPECTRUM name CUTS=0.25,0.5,0.75 /CONTOUR
```

draws cuts at $1/4, 1/2, 3/4$ of the spectrum maximum. The number of specified cuts is unlimited. If the cuts are omitted the last ones are used. As default they are drawn in 10% steps. The isometric representation of two dimensional spectra can be rotated in any direction. The rotation is performed in two steps: a clockwise rotation around the x–axis by an angle "THETA" (Default is 25.0^0) and a clockwise rotation around the new y–axis by an angle "PHI" (Default is -15.0^0):

```
DISPLAY SPECTRUM name THETA=245.0 PHI=45.0 /ISO
```

13.1.1 Digital and Analogous Spectra

GOOSY provides two types of spectrum binnings:

DIGITAL spectra are used to accumulate integer or bit data. Each spectrum bin corresponds to a number of integer values accumulated into that bin, where the number of integer values is given by the binsize. E.g. a digital spectrum in the range of 0 to 3 and a binsize of 1 contains the values $0, 1, 2, 3$. The GOOSY Display shows the center of gravity of a spectrum bin at the mean value of the data accumulated into that bin: The displayed bin range is the mean value of the data accumulated into that bin $\pm$ the half of the spectrum binsize. As indicated in figure 13.1 only the half of the first and last bin is displayed. For digital spectra the displayed bin does not indicate the range of values accumulated into that bin!

ANALOG spectra are used for the accumulation of float variables. Each spectrum bin corresponds to a range of real values accumulated into that bin. The lower limit of the bin is inclusive, the upper limits is exclusive. E.g. an analogous spectrum in the range of 0 to 3 and a binsize of 1 contains the ranges $[0, 1)$, $[1, 2)$, $[2, 3)$. The analogous spectrum has one bin less than the digital spectrum and the upper limits is exclusive. The displayed bin range indicates the range of values accumulated into that bin, as shown in figure 13.1. One bin less is shown than in the digital spectrum.

13.2 Global Display Parameters for Spectra

Sometimes the default display parameters, used by the GOOSY Display are not sufficient for your demand. In that case it is possible to predefine the parameters of the `DISPLAY SPECTRUM` command. E.g. a lower limit threshold for the display and the display modes of the axis are set by:

```
DEFINE DISPLAY SPECTRUM (5,) /LOG/XSQRT
```

Except the "BINFACTOR", "CUTS", "THETA", and "PHI" all other parameters of the `DISPLAY SPECTRUM` command can be set. Successive definitions of global display parameters are cumulative and do not distroy the earlier parameter set. Normally the global display modes are activated and used after their definition. At any time a deactivation and activation is possible:

```
DEFINE DISPLAY SPECTRUM /NOACTIVE ! Deactivate global parameters
DEFINE DISPLAY SPECTRUM /ACTIVE   ! Activate global parameters
```

The global parameters are released or activated for one display command by:

```
DISPLAY SPECTRUM /GLOBAL          ! Activate global parameters
DISPLAY SPECTRUM /NOGLOBAL        ! Suspend global parameters
```

All global parameters are listed by:

```
SHOW DISPLAY GLOBALS /SPECTRUM
```

13.3 Displaying Scatterplots

A scatterplot is used if correlations between two event parameters should be displayed:

```
    DISPLAY SCATTER event.x_axis event.y_axis (0,2048,0,500)
                    XLETTER=Energy YLETTER="Delta E"
```

In that example the coincidences of the two members "x_axis" and "y_axis" in the Data Element "event" are shown. The first parameter is displayed in x–direction (horizontal axis) the second one in y–direction (vertical axis). The range, in which the parameter values vary has to be defined both in x and y. The default range is $(0, 1023, 0, 1023)$. At the x-axis the lettering "Energy" appears, whereas at the y-axis "Delta E" is displayed. If the parameter names and the displayed range are omitted the previously given values are used.

The display gets the scatter data from the analysis. Therefore the scatter plot information has to be passed to the analysis. This is done internally by creating a dynamic list entry in the default dynamic list $SCATTER (for the explanation of dynamic lists see the **GOOSY Data Acquisition and Analysis** manual). If any other list should be used specify the dynamic list name in the display scatter command:

```
    DISPLAY SCATTER event.x_axis event.y_axis (0,2048,0,500) DYN_SCAT=list
```

The analysis recognizes the new dynamic list entry, opens a communication path to the GOOSY Display process and starts the transfer of scatter data to it. The dynamic list has to be specified only in the first display scatter command, it is used in all subsequent commands requiring a dynamic list name.

The dynamic list must be attatched and the analysis must run to see scatter plot points on the screen. The dynamic list should be attached as the last list, to make sure that all referred conditions are executed propperly.

Sometimes the scatter data should be filtered by a condition:

```
  DISPLAY SCATTER event.x_axis event.y_axis (0,2048,0,500) condition object
```

The condition can be specified with or without a condition object. If no object has been defined it is assumed that the condition has been executed by the analysis or by any dynamic list. In that case only the result bit is checked. If an additional condition object has been specified, an entry in the dynamic list is created and the condition check against the specified object (any event parameter) is performed. In any case if the result bit is true the scatter data are sent to the display. If a condition object has been specified, take care that no entry in the dynamic list for the same condition exists, except the condition objects are the same. If this is not the case, the result from the last condition check is used for all accumulations and scatter plots, which could yield to unexpected results. In the analysis the event loops perform the following steps (for details see the **GOOSY Data Acquisition and Analysis** manual):

1. Call unpack routine (event input).
2. Call user analysis routine.

3. Execution of Dynamic Lists.
4. Fill output event into output buffer (event output).

Normally the scatterplot is activated by the `DISPLAY SCATTER` command, but this is suspended by:

```
DISPLAY SCATTER /NOACTIVE
```

The scatterplot can be activated at any time in two modes:

1. in an **asynchronous** mode. The analysis continues when a scatter buffer has been sent to the display. It may happen that more data will be analysed than displayed in the scatterplot. That mode is selected by:

   ```
   START SCATTER /ASYNCHRONOUS
   ```

   This is the default mode for all scatterplots.

2. in the **synchronous** mode. The analysis waits for an acknowledge from the display and continues if the scatter data are accepted by the display. This mode slows down the analysis! All data analysed is displayed in the scatterplot. This mode is activated by:

   ```
   START SCATTER /SYNCHRONOUS
   ```

An active scatter plot can be stopped any time by:

```
STOP SCATTER
```

ATTENTION: **Never stay in menu after displaying scatterplots. Display cannot scatter during menu prompt!**

NOTE after starting a **new** enviroment one should clear the dynamic list previously used for scatterplots from all old entries. This is done by:

```
DELETE DYNAMIC ENTRY SCATTER list * *
```

This command is executed in the Data Base Manager.

13.4 Plotting Scatterplots

If one wants to get a hardcopy of a scatter plot, the easiest way is to use a local copy of the screen. If a hardcopy is not available at the display terminal, one must allocate a plotfile, display the scatterplot, deallocate the plotfile, and send the plotfile to the printer, i.e.:

```
GOOSY> ALLOCATE DEVICE s1 LN03  ! File s1.LN3
GOOSY> DISPLAY SCATTER ....
GOOSY> DEALL DEV s1.ln3
GOOSY> <CTRL>Z
$ PATEX s1.ln3                  ! Print file
```

Note that the default file type is .LN3 and that you must specify the file type in the `DEALLOCATE` command. The plotfile in the above example is sent to the laser printer LN03_A. The LN03 device may be the second or first allocated. If it is the only one, scatterplot files can be produced in batch jobs.

Chapter 14

Pictures

To control a complex detector array the display of single spectra or scatterplots is not sufficient. Therefore the definition of several spectra and/or scatterplot displays can be gathered in pictures, which are Data Elements in your Data Base. Pictures may be composed of up to 64 spectra or scatterplots. Any single picture element, called frame can be modified at any time. Furthermore global display parameters valid for all frames may be defined.

14.1 Creating Pictures

Creating a picture you have to know the total number of involved spectra and scatterplots. Each spectrum or scatterplot is organized in one independent frame. The total number of frames must be known to create a picture:

```
    CREATE PICTURE name 10
```

creates the picture "name" with 10 frames. The number of frames is fixed after the picture creation and could not be changed later. The picture frames are numbered starting with 1 for the upper left frame (see figure on page 221). If you intend to produce scatterplots in the picture you have the possibility to define a main condition for all scatter frames:

```
    CREATE PICTURE name 10 condition object
```

Similar to the scatterplots the condition can be specified with or without a condition object. If no object is declared, it is assumed that the condition has been executed earlier and only the result bit will be checked. If additionally a condition object is given the condition limits will be checked against the object and an earlier check of the same condition is overwritten. If the result bit of the condition is true the scatter data are sent to the GOOSY Display process. An existing picture is deleted by:

```
    DELETE PICTURE name
```

The CREATE PICTURE and DELETE PICTURE commands are executed in the Data Base Manager!

After the creation of a picture all frames are undefined. Before displaying a picture **all** frames have to be pre–set with valid spectra names or scatterplot parameters.

14.2 Modifying Picture Frames

In a single picture frame all informations necessary for the display of that frame are stored. One has to distinguish between spectra and scatter frames:

Spectrum frame For a spectrum frame all parameters of the "DISPLAY SPECTRUM" command, exept "BINFACTOR", "CUTS", "THETA", and "PHI", can be set.

```
MODIFY FRAME SPECTRUM picture frame ....
```

The modify command changes any specified parameter in the "frame" of "picture". The modification of several frames within one modify command is supported if the picture should be filled with members of a one dimensional spectrum array. In that case a range of members in the spectrum array and a range of frames can be specified. Subsequent spectra are put into subsequent frames:

```
MODIFY FRAME SPECTRUM picture 1:4 spectrum(3:6)
```

In this example the spectra "spectrum(3)" to "spectrum(6) are put into the frames 1 to 4!

Additionally it is possible to define a dynamic scaling axis. This is useful if you like to increase or decrease the default GOOSY Display scaling axis by a certain amount. But normally the spectrum contents change when the analysis is active, so no fixed values should be specified in that case. The dynamic scaling factor is used as a factor to the maximum spectrum content. E.g. to increase the scaling axis to twice the spectrum maximum, specify:

```
MODIFY FRAME SPECTRUM picture frame SCALEFACTOR=2.0
```

Scatter frame For scatter frames all parameters of the "DISPLAY SCATTER" command, exept the "DYN_LIST", can be set:

```
MODIFY FRAME SCATTER picture frame ....
```

The modify command changes any specified parameter in the scatter "frame" of "picture".

Beside the main picture condition, the definition of a frame condition is possible with or without a condition object. Scatter data for this frame are shown if both the picture and the frame conditions are true.

The MODIFY FRAME commands are executed in the Data Base Manager!

If frame conditions with a condition object are specified, take care that the same condition never refers to different condition objects!

If a condition object is specified in the picture the condition is inserted in the dynamic list when the picture is displayed. The result of this condition check is used for the scatter display.

Successive frame modifications are cumulative, which means that only the specified parameters are changed, the other values in the frame remains unchanged. Normally it is possible to change a scatter frame into a spectrum frame and vice versa, if no overlays are defined in the picture (see chapter 17.5 on page 239).

Additionally to the frame definition via the modify commands, it is possible to invoke interactive prompting menus. In that mode for each frame a menu is created, which allows to specify all necessary input parameters. This can be used in interactive sessions, to be sure that all frames are properly specified. The interactive prompting menus are created by:

```
CREATE PICTURE name 10 /PROMPT
CREATE PICTURE name 10
```

To create pictures in a command procedure specify:

```
CREATE PICTURE name 10 /NOPROMPT
```

to prevent the interactive inquiry. The definition of the single frames occurs by successive `MODIFY` commands. The parameters set in the picture are listed by:

```
SHOW PICTURE picture /FULL/DATA
```

In the picture and Directory name wildcards are allowed. The switch "/FULL" effects the listing of the spectra and scatter parameters for each frame. With the "/DATA" switch all other specified values are listed.

14.3 Displaying Pictures

If all frames are defined the picture is displayed by a single command:

```
    DISPLAY PICTURE picture
```

The parameters stored in the picture definition are used, creating the display of the single frames. But they can be changed for the current display command, if other parameters are specified, e.g:

```
    DISPLAY PICTURE picture /LIN
```

displays the picture frames with a linear scaling axis. Similar to the "DISPLAY SPECTRUM" command a binning factor can be specified to prevent the automatic display binning (see page 210):

```
    DISPLAY PICTURE picture BINFACTOR=bins
```

If scatter frames are involved in the picture the declaration of a dynamic list is possible if an other list than the GOOSY default $SCATTER should be used. In that dynamic list a dynamic entry is created to activate the preparation of the scatter data in the analysis (see page 213).

```
    DISPLAY PICTURE picture DYN_SCAT=list
```

ATTENTION: **Never stay in menu after displaying pictures containing scatterplots. Display cannot scatter during menu prompt!**.

NOTE after starting a **new** enviroment one should clear the dynamic list previously used for scatterplots from all old entries. This is done by:

```
        DELETE DYNAMIC ENTRY SCATTER list * *
```

This command is executed in the Data Base Mananger.

14.4 Global Display Parameters for Pictures

The global display parameters for pictures are used in the same way as the global parameters for spectra. They can be set and activated by:

```
    DEFINE DISPLAY PICTURE ...   ! specify any parameter
```

Except the "CONDITION", "DYN_SCAT", and "BINFACTOR" all other parameters of the display picture command can be set. Successive definitions of global display parameters are cumulative and do not destroy the earlier parameter set. Normally the global display modes are activated and used after their definition. At any time a deactivation and activation is possible:

```
    DEFINE DISPLAY PICTURE /NOACTIVE ! Deactivate global parameters
    DEFINE DISPLAY PICTURE /ACTIVE   ! Activate global parameters
```

Furthermore the global parameters are released or activated for one display command by:

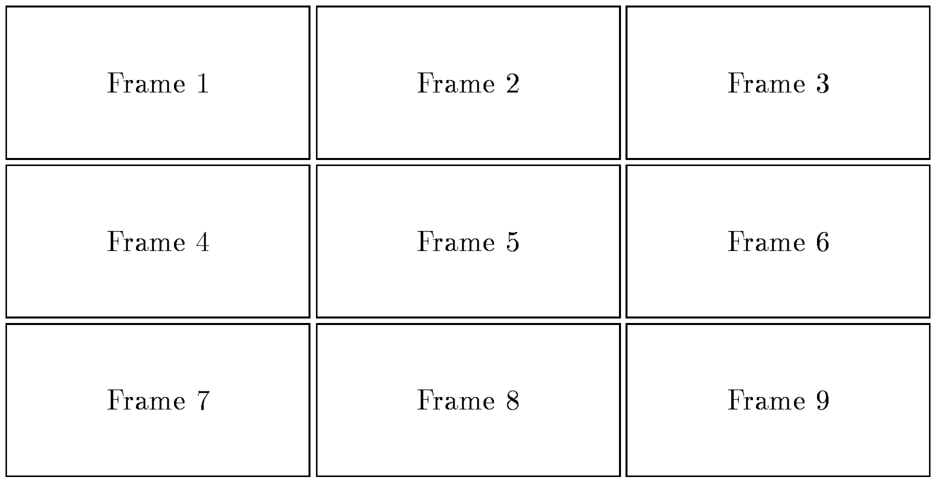


Figure 14.1: Standard Set–Up for Picture with 9 Frames

```
    DISPLAY PICTURE /GLOBAL          ! Activate global parameters
    DISPLAY PICTURE /NOGLOBAL        ! Suspend global parameters
```

All global parameters are listed by:

```
    SHOW DISPLAY GLOBALS /PICTURE
```

14.5 Picture Set–Up

Creating a picture the frames are arranged in rows and columns on the screen and the size of the frames is equal (see figure 14.1). In principle the size of the frames and their position on the screen is arbitrary, with the restriction that an overlap of two frames could yield in unexpected results if cursor inputs are provided. There is one command which allows a redefinition of the picture frames at any time after the creation of the picture. You can specify the number of rows in which the frames should be arranged, that means the vertical extension of all frames is the same. Furthermore for each row the number of frames in that row have to be specified (see figure 14.2):

```
    DEFINE PICTURE SETUP picture 4 3,3,2,1 ! 4 horizontal rows
                                           ! 3 frames in row 1 and 2
                                           ! 2 frames in row 3
                                           ! 1 frame in row 4
                                           ! total of 9 frames
```

Normally the frames in each row are spread over the horizontal screen range. If the size of the frames should be identical specify (see figure 14.3):

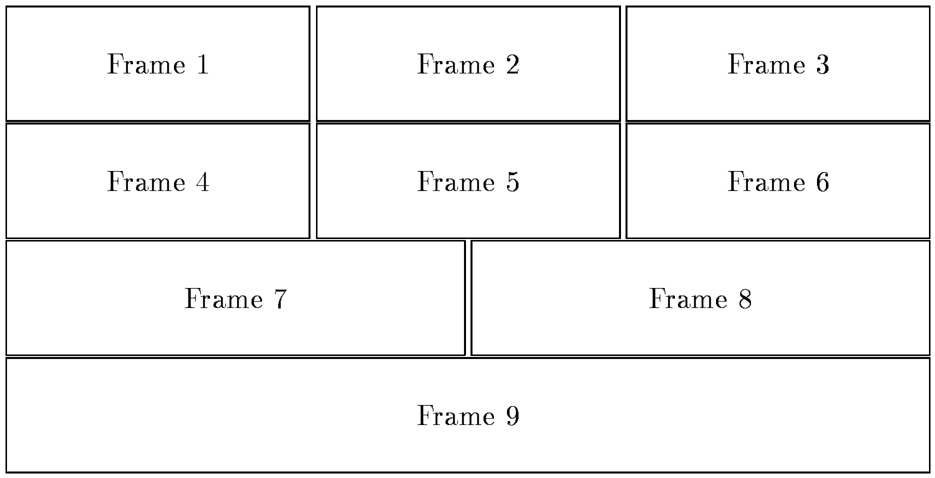


Figure 14.2: User Defined Set–Up for Picture with 9 Frames

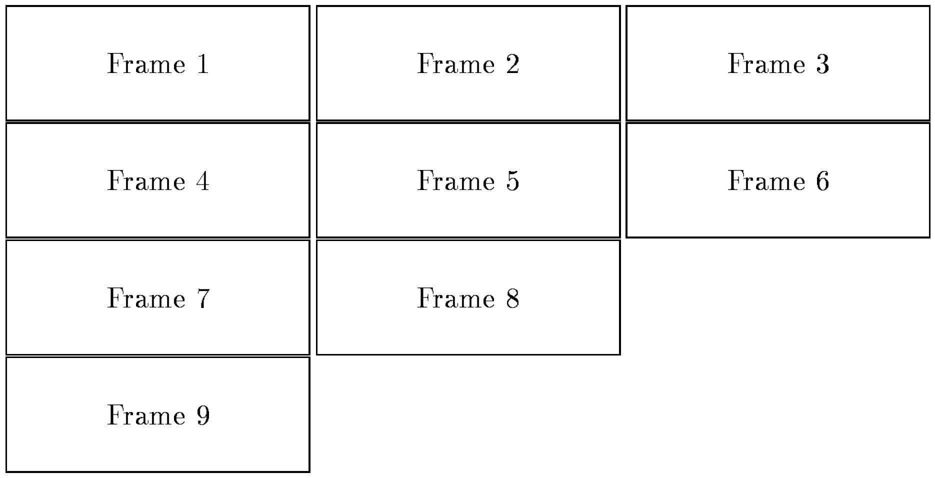


Figure 14.3: Set–up with /EQUAL Switch for Picture with 9 Frames

```
    DEFINE PICTURE SETUP picture 4 3,3,2,1 /EQUAL
```

14.6 Clear Pictures

Normally pictures are used to summarize several spectra or scatterplots which are in a physical relation. And sometimes it might happen that you wish to clear all the spectra arranged in one picture, e.g if analysis parameters are changed. This is done by:

```
    CLEAR PICTURE picture 3       ! Clear spectrum in frame 3
    CLEAR PICTURE picture 1:8     ! Clear spectra in frame 1 to 8
    CLEAR PICTURE picture *       ! Clear spectra in all frames
    CLEAR PICTURE * * /LOG        ! Clear spectra in all frames of all
                                  ! pictures
```

Chapter 15

Special GOOSY Display Concepts

15.1 The GOOSY Display Mode Concept

Discussing the display commands for spectra and pictures, temporary and global display modes have been introduced. But two other parameter sets exist: the defaults provided by the GOOSY Display and the modes specified directly in the commands. Now the priority and correlations of these parameter sets will be described. They are listed in the following with increasing priority:

GOOSY defaults are the values provided by the GOOSY Display to guarantee that all parameters are specified, which are absolutely necessary for the display. *The GOOSY defaults are placed at the lowest priority level.*

Global modes are user defined default values. By default they are active, but a deactivation is possible at any time. A permanent [de]activation is obtained by:

```
DEFINE DISPLAY SPECTRUM /[NO]ACTIVE
DEFINE DISPLAY PICTURE /[NO]ACTIVE
```

A [de]activation for a single command is possible, too:

```
DISPLAY SPECTRUM /[NO]GLOBAL
DISPLAY PICTURE /[NO]GLOBAL
```

For pictures and spectra separate parameter sets exist.

Temporary modes are the values specified in the previous display command. They are identical for spectra and pictures and can be activated in the display commands:

```
DISPLAY SPECTRUM /TEMP
DISPLAY PICTURE /TEMP
```

Command modes are specified in the actual display commands and are available in the next display command as temporary parameters. *The command parameter have the highest priority.*

Parameter values in sets of lower priority are superseded by values of higher priority. If e.g. the global parameter "/LOG" is specified for the scaling axis, the GOOSY Display default "/LIN" is no longer valid. The global value is superseded, if e.g. "/SQRT" is specified in the command. The hierarchy of the GOOSY Display parameter sets is shown in figure 15.1.

15.2 The GOOSY Display Window Concept

For displaying spectra or pictures the following windows are available in the GOOSY Display :

1. The **full window**, which displays the spectrum data in the full spectrum range. It is selected by:

   ```
   DISPLAY SPECTRUM /FULL
   DISPLAY PICTURE  /FULL
   ```

   This full window is the default in the DISPLAY SPECTRUM command.

2. The **specified window**, which displays the spectrum data in the specified range. The specified range are the limits defined in the modify frame commands or directly in the display spectrum command. It is activated by:

   ```
   DISPLAY SPECTRUM LIMITS=range
   DISPLAY PICTURE /SPECIFIED
   ```

   The specified window is the default in the DISPLAY PICTURE command if in the MODIFY FRAME SPECTRUM command limits has been defined.

3. The **last window** is the window of the last display command. Normally, these windows are different for each frame, depending on the actions peformed with the single frames (e.g which region has been expanded). These windows are kept separately for each picture and for one and two dimensional spectra! These windows are indicated in each frame by a # and can be selected by:

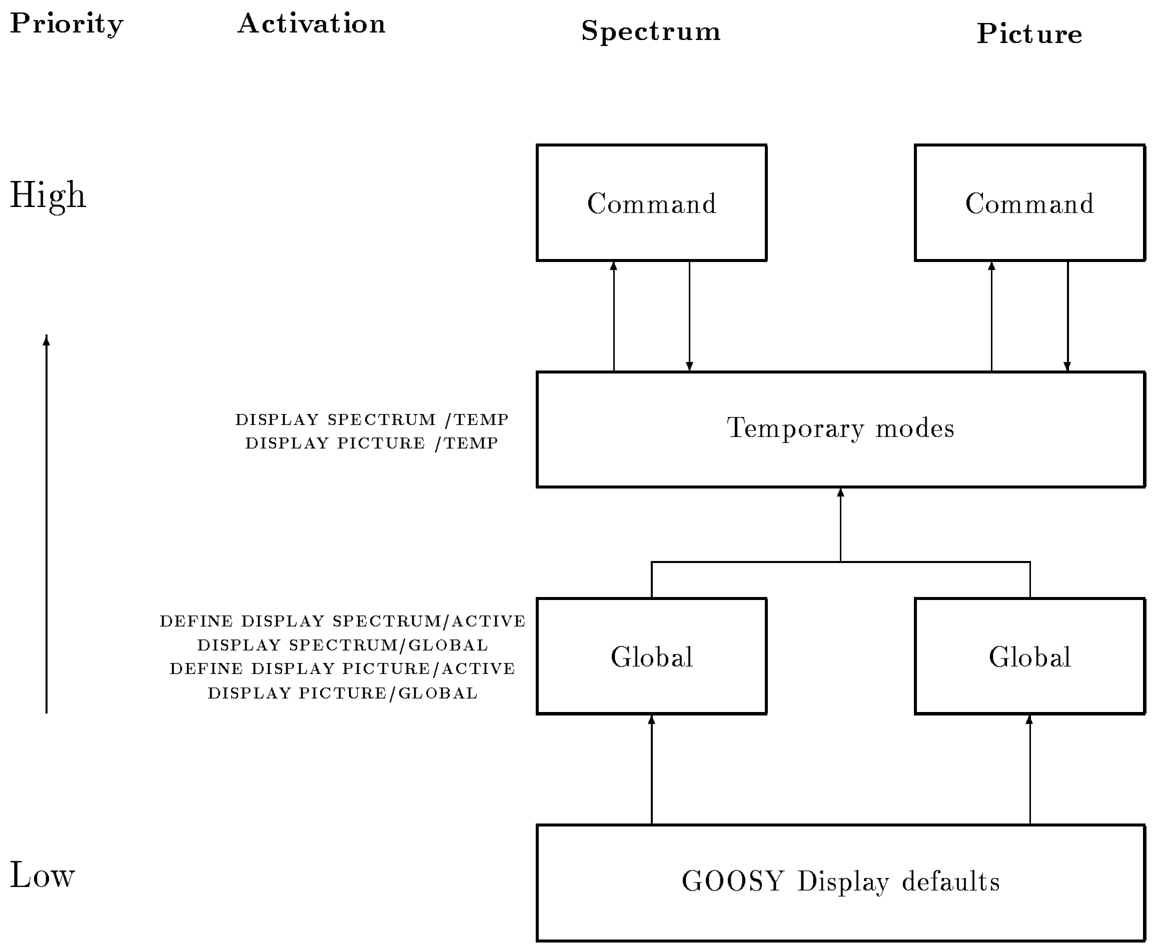


Figure 15.1: Priorities of the Different GOOSY Display Modes

```
          DISPLAY SPECTRUM /LAST
          DISPLAY PICTURE  /LAST
```

The last window is the default in the DISPLAY PICTURE command if in the MODIFY FRAME SPECTRUM command no limits are specified. If the display range of a frame is changed, this range is used in the next display picture command!

4. The **actual window** is the currently displayed range, it can be used for the next display, specifying:

```
          DISPLAY SPECTRUM /ACTUAL
          DISPLAY PICTURE  /ACTUAL
```

In several commands it is posible to specify the window limits directly; e.g.:

```
DISPLAY SPECTRUM LIMITS=window
```

The GOOSY Display provides special window specification possibilities. The window limits are enclosed in brackets and the values are separated by commas. The first two values are reserved for the minimum and maximum values for the x–axis (horizontal direction), the next two values are the limits on the y–axis (vertical direction) for two dimensional spectra:

```
DISPLAY SPECTRUM LIMITS=(xmin,xmax)           ! one dimensional window
DISPLAY SPECTRUM LIMITS=(xmin,xmax,ymin,ymax) ! two dimensional window
```

In all window specifications it is possible to declare only one limit, the upper or lower values can be skipped by an empty input, they are replaced by the highest, lowest possible input:

```
DISPLAY SPECTRUM LIMITS=(5,)     ! Lower display limit is 5
                                 ! Upper limit is upper spectrum range.
DISPLAY SPECTRUM LIMITS=(,400)   ! Display from lower spectrum limit
                                 ! up to 400
DISPLAY SPECTRUM LIMITS=(,,200)  ! Display in full x-range and
                                 ! from 200 till upper spectrum limit
                                 ! in the y-range
```

For other commands, like EXPAND or INTEGRATE, the lowest and highest possible input values are determined by the actual display window! With these window concept is possible to define lower or upper display thresholds which are independent on the limits of the actually displayed spectra.

Chapter 16

Displaying Calibrated Spectra

Displaying spectra in arbitrary units is sometimes not very informative. Normally one is interested in physical units, especially if different spectra should be compared. Therefore the GOOSY Display provides user defined calibration functions calibrate spectra. The calibrations are kept in tables in the Data Base and can be connected to an arbitrary number of spectra. Afterwards a display of the spectrum in calibrated units is possible.

16.1 Creating Calibrations

To display spectra in calibrated units you need a well defined unambigious function. On the other hand, if you prefer to think in calibrated units, the inputs of all spectrum limits should occur in calibrated units. In that case an unambigious inverse calibration function is needed to convert the inputs back into spectrum units. If that is requested, only one method is possible: the calibrations have to be kept in tables, which allow to calibrate and recalibrate each value. GOOSY provides three different types of calibration tables, to fit all requirements:

1. The **linear calibrations** are linear polynoms. For these functions the calibration and recalibration is defined and can be determined exactly. Therefore the polynom parameters are kept in the calibration Data element. They are created by:

   ```
   CREATE CALIBRATION LINEAR name
   ```

2. The **fixed calibrations** are created with a fixed step width for the table entries of the calibrated values. Therefore only a list of calibrated values is kept in the calibration table. The table range is defined by the value of the first table entry, the step width between two entries and the table length. These calibrations should be used for functions which vary slowly. They are created by:

```
CREATE CALIBRATION FIXED name entries
```

Besides the calibration "name" the number of calibrated values ("entries"), stored in the table, must be specified.

3. The **float calibrations** are created with a variable step width for the table entries of the uncalibrated values. Therefore a list of calibrated as well as the corresponding uncalibrated values are kept in the Data Element. These calibrations should be used for functions which vary drastically. They are created by:

   ```
   CREATE CALIBRATION FLOAT name entries
   ```

   Besides the calibration "name" the number of calibrated values ("entries"), stored in the table, must be specified.

16.2 Set Calibration Tables

The calibration tables can be set in different modes:

1. The table entries are directly specified.
2. A list of calibrated and uncalibrated values is given and a polynom fit should be performed. This polynom is used to determine the table entries.
3. A list of calibrated values is given. The corresponding uncalibrated values are specified via cursor inputs. Then a polynom fit is performed.
4. A user module is called, which is used to fill the calibration table.

Sometimes the calibrated values are calculated by a user program and stored in a disk file. If the file is written in a specific format, it is possible to read the required inputs from that file.

The input possibilities for the specification of the calibration tables are different for each type of calibration:

Linear calibration In the linear calibration tables the parameters of a linear polynom are kept. There are different possibilities to determine the polynom parameters:

1. They can be specified directly:

   ```
   SET CALIBRATION LINEAR name "units" PAR=(100.0,0.5) /PARAMETER
   ```

The linear calibration "name" is specified with an offset 100.0 and a polynom factor of 0.5. The calibration description "units" is displayed at the calibration axis, if this calibration has been applied to a spectrum.

2. Another possibility is to determine the polynom parameter by fitting a list of calibrated and uncalibrated values:

   ```
   SET CALIBRATION LINEAR name "units" CALIB=list UNCALIB=list/FIT
   ```

   It is possible to read these two lists of values from a file:

   ```
   SET CALIBRATION LINEAR name "units" file/FILE/FIT
   ```

   The format of the input file is described below on page 233.

3. Sometimes the list of calibrated values is fixed (e.g for calibration sources) in that case this list can be read from file or specified directly and the corresponding uncalibrated values can be prompted by cursor:

   ```
   SET CALIBRATION LINEAR name "units" file/FILE/PROMPT
   SET CALIBRATION LINEAR name "units" CALIB=list/PROMPT
   ```

   After that a polynom fit is performed and the parameters are stored in the calibration Data Element.

Fixed calibration The step width between two successive uncalibrated values is fixed. Therefore the range of the calibration table is defined by the first uncalibrated value and the step width.

1. The corresponding calibrated values can be specified directly or read from a disk file by:

   ```
   SET CALIBRATION FIXED name "units" 100.0 1.5 CALIB=list /TABLE
   SET CALIBRATION FIXED name "units" 100.0 1.5 file /FILE/TABLE
   ```

   The first uncalibrated value for which a table entry exists is 100.0 and the step width is 1.5; these are spectrum units. "/TABLE" means that all table entries are specified and "/FILE" directs the input to "file". The format of the input file is described on page 233.

2. Similar to the setting of the linear calibrations it is possible to specify the parameter of an arbitrary polynom, which is used to calculate the table of calibrated values:

   ```
   SET CALIBRATION FIXED name "units" 100.0 1.5 PARA=(10,2,0.5) /PARAM
   ```

The following polynom is used to determine the calibrated values:

```
cal = 10.0 + 2.0*uncal + 0.5*(uncal**2)
```

3. The polynom parameter can be determined by fitting calibrated and uncalibrated data. The same features as for the linear calibrations are realized:

```
SET CALIBRATION FIXED name "units" 100 1.5 CAL=list UNCAL=list/FIT
SET CALIBRATION FIXED name "units" 100 1.5 file /FILE/FIT
SET CALIBRATION FIXED name "units" 100 1.5 CAL=list/FILE/PROMPT
SET CALIBRATION FIXED name "units" 100 1.5 file /FILE/PROMPT
```

4. Additionally the calibration table can be set by calling a user module, which is linked into a sharable image:

```
SET CALIBRATION FIXED name unit 100.0 1.5 MOD=module IMAG=image/MOD
```

The user module is called with the following parameter list:

```
L_sts = module(R_start,R_step,L_entries,RA_table)
               ! R_start - start value of table
               ! R_step  - Step width between two entries
               ! L_entries - Number of table entries
               ! RA_table -  Array which contains the final
                             calibration values.
```

Float calibrations The step width between two successive uncalibrated values varies. In that case all uncalibrated and calibrated table entries have to be stored.

1. The table can be set by a user module with the following command:

```
SET CALIBRATION FLOAT name "units" MODULE=module IMAGE=image
```

Your module has to be linked into the sharable image "image" and is called with the following parameter list:

```
status=program(L_entries,RA_uncal,RA_cal);
   ! L_entries    Number of table entries
   !              BIN FIXED(31) Input
   ! RA_uncal     Array with uncalibrated values
   !              (L_entries) BIN FLOAT(24) Output
   ! RA_cal       Array with calibrated values
   !              (L_entries) BIN FLOAT(24) Output
```

2. Or it is possible to read the total calibration table (the lists of calibrated and uncalibrated values) directly from the file "file":

   ```
   SET CALIBRATION FLOAT name "units" file /FILE
   ```

Take care that in the calibration table for each uncalibrated value only one calibrated value exists, and vice versa. If this is not the case the result obtained displaying a calibrated spectrum is unpredictable.

After the calibration tables has been set, the unambigiuty of the inverse calibration is proofed and a warning message is created if the calibration is not correct. The calibration function can be displayed by:

```
DISPLAY CALIBRATION name
```

This is a convient way to control the correctness of the calibration.

Calibration Input File Format

Calibration files are ASCII text files. The format of the files is very simple: An exclamation point (!) in the first column indicates a comment line. In each record one or two input values are expected depending on the input mode:

/FIT Two input values, separated by comma or blanks, are expected in each line. The first one is the uncalibrated value, the second one the corresponding calibrated value.

/PROMPT For each line one calibrated input value is expected.

/TABLE For a fixed calibration one calibrated value is expected in each line. For the float calibrations a list of uncalibrated and calibrated values are read.

E.g. for the "/FIT" input mode the file looks like:

```
!
!    uncalibrated          calibrated  value
!
     500.0                    800.0
     700                      1020.0
!  Another comment line
     1020,                    2000.0
     1040,2100
!
!   end of file
!
```

Linking Modules into a Sharable Image

Dynamically loaded user modules have to be linked into a sharable image. This can be done by the DCL command `LSHARIM`:

```
  $ LSHAR module image
```

One or several modules can be linked together into the specified sharable image. If the modules uses internally global variables (COMMON blocks in FORTRAN or EXTERNAL variables in PL/1), they must be defined explicitly and must be placed into an unshared section in the image. This is done by `LSHARIM` if global parameters are specified:

```
  $ LSHAR module image /GLOBAL=variables
```

For the dynamic load of the sharable image a logical name has to be assigned to it:

```
  $ LSHAR module image /SHARELOG=name
```

16.3 Calibrating Spectra

Up to now the created calibrations are independent Data Elements in your Data Base. But they should be connected to one or several spectra:

```
    CALIBRATE SPECTRUM spectrum calibration
```

The connection between spectra and calibrations is established creating links between the Data Elements (see the **GOOSY Data Management** manual).

Up to now only one dimensional spectra can be calibrated.

It is possible to connect one calibration to several spectra, but it is not possible to connect one spectrum to several calibrations. Furthermore the calibration of a spectrum can be changed any time by disconnecting it from its current calibration and then connecting it to another one. The decalibration of a spectrum is performed by

```
    DECALIBRATE SPECTRUM spectrum
```

the "spectrum" is disconnected from its current calibration. If you want to delete the spectrum, it must be decalibrated first. A list of all spectra connected to a calibration can be obtained by:

```
    SHOW CALIBRATION calibration /LINKS /TABLE
```

The switch "/LINKS" shows all established links and "/TABLE" lists the total contents of the calibration table.

16.4 Displaying Calibrated Spectra

When the calibrations are made up calibrated spectra can be displayed. In the GOOSY Display two calibration modes are implemented:

1. A calibrated axis is drawn below a spectrum shown in the original coordinate system. Using non–linear calibrations the distance between two neighbouring tics varies. This mode is activated by:

   ```
   DISPLAY SPECTRUM /CALAX
   DISPLAY PICTURE /CALAX
   ```

2. The spectrum is drawn in calibrated units. Then the size of the displayed spectrum bins vary. This mode is activated by:

   ```
   DISPLAY SPECTRUM /CALSP
   DISPLAY PICTURE /CALSP
   ```

In any case the uncalibrated axis is displayed, too. This can be suppressed by:

```
DISPLAY SPECTRUM /CALAX/NOCHAN
DISPLAY PICTURE /CALAX/NOCHAN
DISPLAY SPECTRUM /CALSP/NOCHAN
DISPLAY PICTURE /CALSP/NOCHAN
```

Chapter 17

Often Used Display Commands

In this chapter some useful commands are presented. The most common actions are the integration of a spectrum range or the expansion of spectra and scatterplots.
Comparing spectra in comfortable manner can be done by overlays. Definable, linear transformations are applied to the overlayed spectra in x and y–direction.
The standard GOOSY Display keeps all graphical output in memory in a device independent form. Therefore it is possible to plot the currently active picture on any supported plotter.

17.1 Expand

Sometimes the displayed range in one or several frames is not sufficient and you like to change the display limits. This is done by the "EXPAND" command. You have the possibility to specify the new range graphically by:

```
EXPAND                 ! without any parameter
```

The cursor appears at the GOOSY main device (see chapter 12.1 on page 206) and the new display range can be specified in any frame. The corresponding frame is deleted from the display screen and redrawn within the specified limits. For all terminals, which support the erasure of local display parts, the other frames are not affected by that operation. For dump terminals (like TEKTRONIX 4014) the whole screen is erased and redrawn. If the expansion limits are known, they can be specified directly in the command:

```
EXPAND (100,567)           ! for one dimensional spectra
EXPAND (100,567,234,1056)  ! for two dimensional spectra or
                           ! scatter plots.
```

In that examples, the cursor appears, if more than one frame is displayed, to select the frame which should be used. Furthermore it is possible to expand more than one frame by:

```
EXPAND (100,500) 2:4       ! expand frame 2 to 4
EXPAND (100,500) *         ! expand all frames
EXPAND FRAME=*             ! the cursor appears to specify the limits
```

If calibrated spectra are in the selected frames the new display limits can be specified in calibrated or uncalibrated units:

```
EXPAND (100,567) /CHAN      ! limits are uncalibrated units
EXPAND (100,567) /CALIB     ! limits are calibrated units
```

17.2 Integrate

The syntax of the "INTEGRATE" command is similiar to the "EXPAND" command. Limits can be specified directly or graphically and in calibrated or uncalibrated units. Furthermore the integration of spectra in one or in several frames is possible. Some examples:

```
INTEGRATE                        ! cursor input
INTEGRATE FRAME=frame            ! cursor appears in frame
INTEGRATE (100,200) *            ! integrate spectra in all frames.
```

17.3 Sum Up Spectrum

The `INTEGRATE` command provides graphical inputs. If the limits of the window which should be integrated are known you can use `SUMUP` instead of `INTEGRATE`:

```
SUMUP SPECTRUM spectrum (100,200)
```

It is possible to define more than one window by:

```
SUMUP SPECTRUM spectrum (100,200,400,500)
```

Similiar to the `INTEGRATE` command the window limits can be specified in original spectrum units or in calibrated units. *The* `SUMUP SPECTRUM` *command is executed in the Data Base manager!.* No display of the spectrum is needed.

17.4 Refresh

Sometimes the current picture should be re–displayed to get rid of superfluous information. E.g if scatterplots are in the picture you can erase the scatterpoints, without another `DISPLAY PICTURE` or `DISPLAY SCATTER` command and without creating a dynamic entry, which takes some time. This is done by:

```
REFRESH 5               ! Erase frame 5 and redisplay it
REFRESH *               ! Redraw all frames
```

If spectrum frames are refreshed, the old spectrum data are displayed. To get the new spectrum contents, specify:

```
    REFRESH 5 /UPDATE
    REFRESH * /UPDATE
```

17.5 Overlays

If spectra or pictures are displayed, overlays of one- and two dimensional spectra are possible:

```
    OVERLAY spectrum FRAME=frame
```

The frame has to be specified if more than one frame is displayed. The overlayed spectrum can be shifted and scaled in x and y–direction:

```
    OVERLAY spectrum FRAME=frame TRANS=(xfac,xoff,yfac,yoff)
    OVERLAY spectrum FRAME=frame
            TRANS=(1.0,0.0,2.0,100.0) ! Shift in y-direction by 100
                                      ! Factor in y-direction: 2.0
    OVERLAY spectrum FRAME=frame
            TRANS=(,,2.0,100.0)       ! Shift in y-direction by 100
                                      ! Factor in y-direction: 2.0
    OVERLAY spectrum FRAME=frame
            TRANS=(0.5,-30)           ! Shift in x-direction by -30.0
                                      ! Factor in x-direction: 0.5
```

Sometimes the transformation factors are unknown. To spare you the calculation of the transformation an automatic adjustment is implemented, which is called by:

```
    OVERLAY spectrum FRAME=frame /ADJUST
```

In that mode the cursor appears twice to select two points in the original spectrum. Then the spectrum, which should be overlayed is drawn and the cursor appears again to inquire two points. These cursor inputs define a linear transformation applied to the overlayed spectrum.

Besides the overlay of spectra the GOOSY Display provides overlays of several scatter parameters:

```
    OVERLAY XPARA=event.x_para YPARA=event.y_para FRAME=frame /SAVE
```

The defined parameters are scattered additionally into the specified frame. The `/SAVE` switch is necessary to save the scatter parameter in the data element of the actually displayed picture data element and to initiate the update of the SCATTER entry in the dynamic list, which enables the analysis to create scatter buffers containing the additional parameter set. The scatter points are displayed in another color, if that is possible. If `/SAVE` is not specified no update of the SCATTER entry in the dynamic list is performed and therefore the additional parameters are not displayed.

If you know that overlays of different spectra are necessary, you can define a set of overlays for each existing picture. These overlay definitions are kept in the picture definition in your database and can be displayed by a single command:

```
OVERLAY             ! without any parameter
```

Define your overlays by:

```
CREATE OVERLAY picture frame spectrum        ! for spectrum frames
CREATE OVERLAY picture frame XPARA=x YPARA=y ! for scatter frames
```

All parameters of the "OVERLAY" command can be specified in the "CREATE OVERLAY" command, the syntax of both commands is nearly identical. If overlayed spectra should be shifted into y–direction you run into problems, because the scaling axis of the original spectra changes if an analysis is active. Therefore a dynamic shift can be specified, which is used as a factor to the actual maximum spectrum contents. An overlay of "spectrum" in frame "2" of "picture" shifted by half of the maximum of the original frame spectrum is defined by:

```
CREATE OVERLAY picture 2 spectrum DYNSHIFT=0.5
```

If the count rate in the overlayed spectrum exceeds the maximum of the original frame spectrum it is possible to generate a dynamic scaling axis (see page 218):

```
MODIFY FRAME SPECTRUM SCALEFACTOR=2.0
```

The scaling factor and the dynamic shift allow an optimal definition of shifted overlayed spectra, independent of the actual spectra maxima.

If the overlays created for a single frame or for the total picture are no longer needed, they can be deleted by:

```
DELETE OVERLAY picture 3   ! overlays of frame 3 are deleted
DELETE OVERLAY picture 2:5 ! overlays of frame 2 to 5 are deleted
DELETE OVERLAY picture  *  ! all overlays are deleted
```

17.6 Updating Frames

The spectrum accumulation can be controlled if in a specified time the new spectrum contents are drawn across the original spectrum. To update all frames on the screen, declare an update time in the display commands:

```
DISPLAY SPECTRUM UPDATE=10    ! Update of spectrum every 10 sec.
DISPLAY PICTURE UPDATE=2      ! Update of all frames every 2 sec.
```

The update of definite frames is performed by:

```
    UPDATE FRAMES 5,4,1 10
```

The spectra in frames 1, 4, 5 are drawn every 10 seconds. An arbitray number of valid frames can be specified.

The update of frames is canceled by any "DISPLAY SPECTRUM" or "DISPLAY SCATTER" or "DISPLAY PICTURE" command.

17.7 Plotting Pictures

The standard GOOSY Display (see page 247) keeps the most graphical output in a device independent format in a workstation independent segment storage (see the **GKS Primer** manual). This storage is kept in the memory or in a disk file if the required memory gets too large. But there are two exceptions: scatter data and the data obtained by the picture update (see chapter 17.5 on page 240) are not stored, because it is unpredictable how many data are produced with scatterplots and frame updates. To spare disk space these outputs are not kept in the device independent format.

The segment storage allows the output of all stored graphical data on any other device, especially a printout on plotter; e.g:

```
    PLOT PICTURE sys$ln03_c ln03 2
```

The actual picture is printed on plotter queue "SYS$LN03_C". The plotter is a LN03–PLUS laser printer and 2 copies are printed.

If there are some scatter frames on the screen they remain empty on the plot, if it is produced with the "PLOT PICTURE" command. To plot the scatter data you have to allocate a plotter or a metafile to save all graphical outputs (see chapter 12 on page 205) on file. After the deallocation of the plotter the produced files can be printed by:

```
    PLOT PLOTFILE file queue copies
```

The plotfile "file" (wildcards are supported) is plotted on "queue" in "copies" versions. Metafiles are printed by:

```
    PLOT METAFILE file queue type
```

The "file" is interpreted and converted in device dependent sequences of a device of the specified "type" and printed out on the specified "queue". The metafile can be sent to the IBM for plotting:

```
    PLOT METAFILE file IBM::   RP01  ! output on RP01: Large Benson
    PLOT METAFILE file IBM::   RP02  ! output on RP02: Small Benson
    PLOT METAFILE file IBM::   VP01  ! output on VP01: Vector plotter
```

17.8 Setting and Displaying Window Conditions

Condition windows are set with the REPLACE CONDITION WINDOW and SET CONDITION WINDOW commands. The difference between REPLACE and SET is that the REPLACE command provides cursor inputs and is therefore available in the GOOSY Display . The SET command requires the window limits and is executed in the GOOSY Database Manager.

A window condition can be multidimensional, therefore it is necessary to specify the dimension in which the condition limits should be set:

```
REPLACE CONDITION WINDOW condition limits 3   ! set limits in dimension 3
REPLACE CONDITION WINDOW condition limits 5:7 ! set dimension 5 to 7 with
                                              ! the specified limits
REPLACE CONDITION WINDOW condition limits *   ! set all dimensions
```

These examples are identical with the SET CONDITON WINDOW command! The specification of the limits is optional in the REPLACE command. If they are omitted the cursor appears and a graphical input is possible. Furthermore it is possible that the limits in one or several members of a condition array should be set:

```
REPLACE CONDITION WINDOW cond(1)   DIM=3   ! set COND(1) in dimension 3
REPLACE CONDITION WINDOW cond(3:5) DIM=5:7 ! set COND(3) to COND(5)
                                           ! in dimension 5 to 7
REPLACE CONDITION WINDOW cond(*)   DIM=*   ! set all members of COND
                                           ! in all dimensions
```

In the examples above the cursor appears and the window limits have to be specified graphically. These inputs are possible in the SET command, too, with the restriction that the limits are then required. In the REPLACE command the input of the window limits in calibrated units is supported by:

```
REPLACE CONDITION WINDOW cond(1) DIM=3/CALIB
```

If the condition windows should be set in two dimensional spectra, it is possible to specify the axis at which the graphical input should occur:

```
REPLACE CONDITION WINDOW cond(1:5) DIM=*/YAXIS
REPLACE CONDITION WINDOW cond(1:5) DIM=*/XAXIS
```

The verification of the condition windows is possible with the DISPLAY CONDITION command. Dimension "5" of the condition window "condition" in frame "2" is displayed by:

```
DISPLAY CONDITION condition FRAME=2 DIMENSION=5
```

Further input possibilities are:

```
    DISPLAY CONDITION cond(1:5) FRAME=1 DIM=3:5
    DISPLAY CONDITION cond(*) FRAME=1 DIM=3:5
    DISPLAY CONDITION cond(*) FRAME=1 DIM=*
    DISPLAY CONDITION cond(*) FRAME=2 DIM=* /DISTRIBUTE
```

The "/DISTRIBUTE" switch distributes the single members of a condition array to subsequent frames, starting at the specified frame. In the above example COND(1) is shown in frame 2, COND(2) in frame 3 etc. Furthermore the outputs can be directed to the y–axis by:

```
    DISPLAY CONDITION cond(1:5) FRAME=1 DIM=3:5 /YAXIS
    DISPLAY CONDITION cond(*) FRAME=2 DIM=* /DISTRIBUTE/YAXIS
```

17.9 Setting and Displaying Polygons

Polygons are set with the `REPLACE POLYGON` command. The polygon data can be specified in a list of x- and y-values:

```
  REPLACE POLYGON polygon XPOINTS=list YPOINTS=list
```

The number of x and y values have to be the same, if specified. If no points are specified a graphical polygon editor is envoked, which allows to delete, insert, or append new points in the polygon. The polygon must be displayed in that case. The following help appears if the polygon editor is activated:

```
    GOOSY Polyline editor is active
    To append or overstrike point press  : SPACE
    To stop editor input press           : RETURN
    To skip one point backward press     : B or b
    To skip one point forward press      : F or f
    To delete current point press        : D or d
    To delete backspaced point press     : <BS>
    To toggle insert mode press          : I or i
```

The insert mode is indicated at the right bottom on the screen. These inputs occur at the graphical input device. A polygon, which has been replaced earlier, can be modified with this editor. In that case the polygon points have to be displayed in an existing frame, which must be specified:

```
  DISPLAY POLYGON polygon 3   ! show existing polygon points in frame 3
  REPLACE POLYGON polygon 3   ! Edit polygon in frame 3
```

Points in an existing polygon are deleted and can be set again by:

```
  REPLACE POLYGON polygon /DELETE
```

If a polygon is specified which does not exist, it will be created. The number of polygon points can be increased at any time, the size of the polygon data element is increased if necessary.
The correctness of the polygon data can be controlled by:

```
DISPLAY POLYGON polygon frame       ! the contour of the polygon is shown
DISPLAY POLYGON polygon frame /FILL ! the interior of the polygon is filled
```

17.10 Projection of Two Dimensional Spectra

The GOOSY Display provides the possibility to project two dimensional source spectra to any dimension. Furthermore it is possible to specify a projection window to restrict the range of the source spectrum, which should be projected to the specified dimension. The result is kept in a specified target spectrum. If the target does not exist, it will be created with the same attributes as found in the projection dimension of the source spectrum. The syntax of the command is:

```
PROJECT source target window dimension
PROJECT source target (100,400) 2 ! Project window at x-axis 100,400
                                  ! Project to the y-axis.
PROJECT source target   *  1      ! Full y-range of source is projected
                                  ! to x-axis.
```

If the window limits are omitted the cursor appears for a graphical input.

If the target spectrum exists arithmetic operations with the target and the result of the projection are supported:

```
PROJECT source target DIM=2 /ADD   ! Add projection to target
PROJECT source target DIM=2 /SUB   ! Subtract projection from target
PROJECT source target DIM=2 /CLEAR ! Clear target spectrum first
```

Beside the specification of a simple projection window it is possible to specify a polygon, which should be used to define a projection region:

```
PROJECT source target WINDOW=polygon /POLYGON
```

The switch "/POLYGON" suggests that the specified window is not interpreted as a pair of limits, but as the name of a polygon! All spectrum bins for which the middle of the bin is inside the polygon are taken into account during the projection.

17.11 Fitting Spectra

In the GOOSY Display it is possible to fit one dimensional spectra by polynomial and gaussian peaks. As default the data displayed in the specified frames are used for the fit. A polynomial of an arbitrary order is fitted to your data by:

```
FIT SPECTRUM 2 5 /BACKGROUND
```

The data of the spectrum in frame 2 are used and a polynomial of a power of 5 is fitted to the data. The cursor appears to select up to 10 data ranges taken into account for the fit. Gaussian peaks are fitted by:

```
FIT SPECTRUM 1 /GAUSS
```

For each peak start values for the peak–position as well as the peak–width have to be specified by cursor input. In many cases the peaks have similiar peak–widhts; then it is possible to specify the width just once

```
FIT SPECTRUM 1 /GAUSS/SAMEWIDTH
```

In both examples the data of the spectrum in frame 1 are used!

Furthermore it is possible to combine gaussian peaks with a polynomial background:

```
FIT SPECTRUM 2 5 /BACKGROUND/GAUSS/SAMEWIDTH/SHOW
```

The "/SHOW" switch effects that the results of all iterations are shown. As default the fit uses the whole data as displayed actually on the screen, but the data region can be expanded or enclosed by specifying a data window:

```
FIT SPECTRUM 2 5 100,600/BACKGROUND/GAUSS
FIT SPECTRUM 2 5 ?/BACKGROUND/GAUSS
```

In the first example the data between the channels 100 and 600 are used for the fit. In the second example the data region to be taken into account during the fit is specified via cursor input.

The data are normally wheightened by their statistical errors. Fitting gaussian peaks this yields in an adjustment to the peak flanks. For data with bad statistics or none gaussian peaks the result may be unsatisfying. Better results are obtained without error weightening:

```
FIT SPECTRUM 2 5 /BACKGROUND/GAUSS/NOERROR
```

17.12 Defining the Frame Setup

The GOOSY Display provides the possibility to set some parameters, which manipulate the layout of the frames. The number of tics at the axis can be increased or decreased, the text fonts can be changed, the size of the spectra in the frames can be increased, the number of channels, which should be displayed for two dimensional spectra is variable, and you can select weather isometric plots will display a grid or not:

```
DEFINE FRAME SETUP tic_number text_font linewidth xyratio channels
                   /[NO]GRID /[NO]INFO
```

The switch "/NOINFO" yields to a larger representation of the spectra, but in that mode no space is available to display detailed spectrum informations. Nice pictures, for any kind of representation, can be produced by increasing the linewidth of all polylines, or by changing the format of the whole picture by the "xyratio".

Chapter 18

Different Display Versions

The GOOSY Display can be activated as a normal image or as a GOOSY process:

```
$ MDISP                          ! standalone display
$ GOOSY CREATE PROCESS DISP $DSP ! creates a GOOSY process
```

Scatterplots and frame updates are not possible if the display is activated as a normal image. The scatterplot and the update are based on the communication mechanism provided by the normal GOOSY environment, which is available only for a GOOSY process.

For scatterplots and frame updates activate the GOOSY Display as a GOOSY process.

As mentioned above the GOOSY Display saves most of the graphical data in a separate workstation independent storage (see page 241). The management of this storage consumes CPU–time. Therefore a display version with a deactivated segment storage is available. This version needs about 60% of the CPU–time of the standard display version. Using the fast display version the functionality of the GOOSY Display is reduced:

1. The "PLOT PICTURE" is not available.
2. The "SAVE DISPLAY" command does not work.
3. The "REFRESH" command works only as

   ```
   REFRESH */UPDATE
   ```

4. The "EXPAND" command works properly only if devices with a local deletion facility are allocated. If e.g. a plotter is active the screen is cleared and the whole picture will be redrawn.

The two display versions are activated by:

```
    SET DISPLAY MODE/STANDARD  ! standard version
    SET DISPLAY MODE/FAST      ! fast display version
```

At any time it is possible to switch from one version to the other. If you work with the fast version and you decide to produce plots, activate the standard version, display a new picture, and then the `PLOT PICTURE` command will work!

The fast version should be used if you run GOOSY on a MICRO–VAX or a VAX 750. On the VAX 8600 an effect is seen if the CPU is occupied by more than 50%.

Chapter 19

Display Glossary

Calibration Calibrations are tables describing the dependence between the original spectrum units and a calibrated units. Each spectrum can be connected to a calibration and a display of the spectrum in calibrated units is possible.

CONDITION In contrast to SATAN, GOOSY conditions are independent of spectra. Besides the multi window conditions which are similar to SATAN analyzer conditions, GOOSY provides window-, pattern-, composed-, polygon- and userfunction-conditions. Each condition has counters associated for true/false statistics. Conditions can be executed in a Dynamic List or by the $COND macro in an analysis routine. Each condition can be used as filter for spectrum accumulation or scatter plots.

CONNECT A calibration can be connected to any number of spectra with the GOOSY command `CALIBRATE SPECTRUM`.

Device All graphical outputs occur at ouput devices; interactive terminals or plotters. In GOOSY up to 10 different devices can be handled at once.

Data Base A Data Base is located in a file and has a Data Base name. It is recommended to use the same name for the file and the Data Base. The file type should be .SEC. A logical name may be defined for the Data Base name. To activate a Data Base it must be mounted. It is dismounted during a system shutdown or by command. If a Data Base runs out of space, it can presently NOT be expanded.

Data Base Directory Similar to a VMS disk, GOOSY Data Bases are organized in Directories. They must be created.

Data Base Pool The storage region of a Data Base is splitted in Pools. All Data Elements are stored in Pools. A Pool can be accessed by a program with READ ONLY protection or with READ/WRITE protection. Pools must be created. They are automatically expanded if necessary, up to the space available in a Data Base.

Data Element A Data Element is allocated in a Data Base Pool. Its name is kept in a Directory. Data Elements can be of atomic Types (scalars or arrays), or of the structure Type (PL/1 structures). Besides the data structure a Data Element can be indexed (one or two dimensional). Such Data Elements are called name arrays. Each name array member has its own data and Directory entry.

Data Element Type GOOSY Data Elements can be PL/1 structures. The structure declarations must be in a file or text library module. They are used to create a Data Element Type in the Data Base and can be included in a program to access the Data Element.

Dynamic List A Dynamic List has several Entries, each specifying an action like condition check or spectrum accumulation. It is executed for each event in the analysis program. The Entries are added or removed by commands even without stopping the analysis.

Dynamic List Entry An Entry in a Dynamic List keeps all information to execute an action. For example, an accumulation Entry contains the spectrum name, an object and optional a condition and an increment parameter.

Global modes are user defined display defaults. For spectra and pictures different parameter sets exist.

Metafile The metafile is a device independent plotfile. This metafile can be redisplayed at any interactive ot non–interactive output device.

Picture A Picture is a complex display. A picture is a set of up to 64 frames with spectra and/or scatterplots. Once created and specified they remain in a Data Base independent of programs. They are displayed by `DISPLAY PICTURE` command. Pictures are composed of frames.

Picture Frame Each frame is a coordinate system for a spectrum or scatter plot. Up to 64 different frames may inserted to a picture.

Scatter Plot The GOOSY display component can display any pairs of Data Element members event by event in scatter plot mode (live mode). Several scatter plots can be displayed on one screen (pictures). Scatter plots are executed in Dynamic Lists and may be filtered by conditions.

Spectrum A GOOSY spectrum differs from a SATAN analyzer in that there are no windows or conditions associated. A spectrum can be filled in a Dynamic List Entry or in an analysis routine by macro $ACCU. GOOSY distinguishes between `DIGITAL` spectra for the accumulation of integer data and `ANALOG` spectra to accumulate float variables.

Temporary modes are the display modes specified in the last `DISPLAY SPECTRUM` or `DISPLAY PICTURE` command.

Part V

GOOSY Data Acquisition and Analysis

Chapter 20

GOOSY Environment

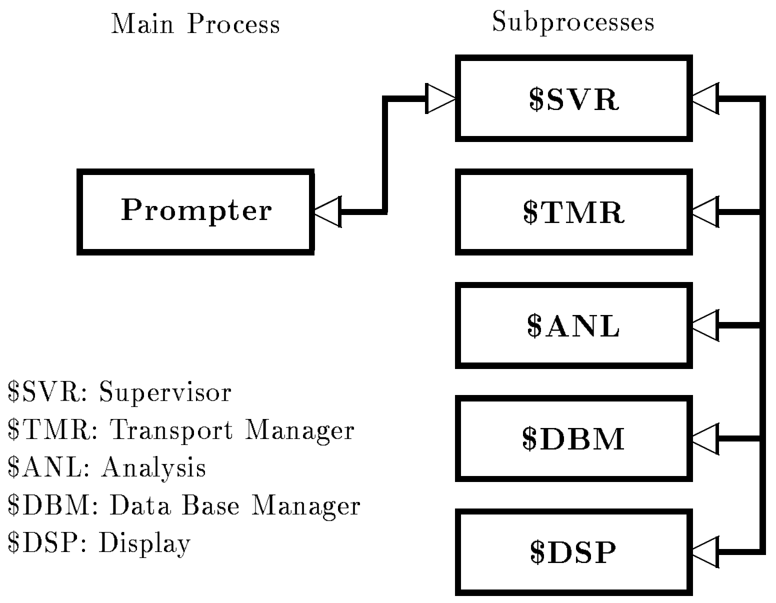


Figure 20.1: GOOSY components shown as VMS processes.

20.1 The GOOSY Environment

GOOSY is composed of several programs. These programs run in parallel. The communication between the programs is done via DECnet or mailboxes. All programs may access data in the Data Base which is common to all programs. In terms of VMS, the programs run in subprocesses as shown in fig. 20.1. In GOOSY, the main process and the subprocesses are called *environment*. There is one environment per terminal. The programs running in an environment are called *components*. GOOSY components are programs executing commands. They get the commands from a prompter program. Thus several components can be controlled from one terminal. Each environment has a supervisor program to dispatch messages between the prompter program and the components. The prompter hibernates until it gets an acknowledge message from the supervisor. Therefore the subprocess executing the command can use the terminal for I/O. See section 20.1.3 to create a GOOSY environment.

20.1.1 The GOOSY Prompter

The GOOSY prompter can only be used after an environment was created! The GOOSY prompter is entered by the DCL command `GOOSY`:

```
   $ GOOSY
it prompts with
   SUC:GOOSY>
```

Now you can enter any GOOSY command. You leave the GOOSY prompter by typing `CTRL` Z. Single GOOSY commands can be given under DCL by a preceding `GOOSY` or just `G`. This allows to execute all GOOSY commands as DCL commands or in DCL command procedures which means also in batch jobs. Vice versa, a single DCL command can be executed by the GOOSY prompter:

```
$ G SHOW TP KEYPAD              !Show keypad layout
$ GOOSY                         !Enter GOOSY prompter
SUC:GOOSY> $ DCL "DCL-command"  !Execute DCL command
SUC:GOOSY> <Ctrl>Z              !Leave GOOSY prompter
$
```

Note, however, that "DCL-command" executes in a separate (spawned) temporary process. The GOOSY prompter menu is displayed by pressing the `NEXT SCREEN` key of your terminal key board.

The GOOSY prompter interprets some special function keys. The definitions are made in GOO$EXE: INI_TP0.COM. A help setup is displayed by the `KP_PF2` key. Figure 20.2 on page 256 shows the keypad layout.

Now, entering the GOOSY prompter, the menus for the different components are entered by:

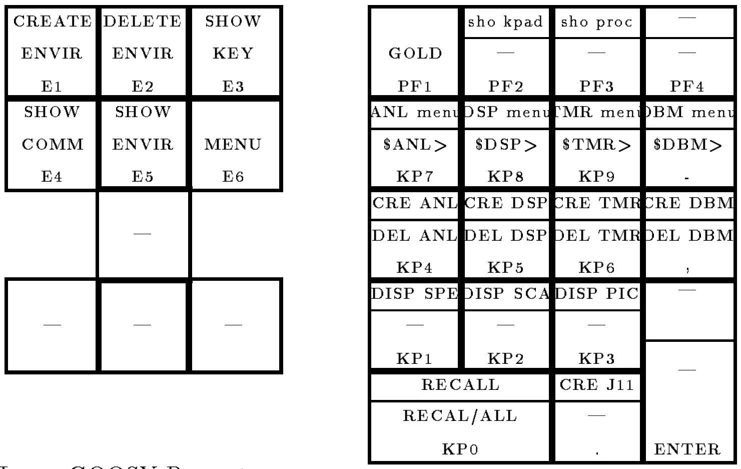
```
<KP_7>     ! $ANL : Analysis commands menu
<KP_8>     ! $DSP : Display commands menu
<KP_9>     ! $TMR : Transport Manager commands menu
<KP_MINUS> ! $DBM : Data Base Manager commands menu
```

Similar, single components can be deleted and created by special keys:

```
[<PF1>]<KP_4>     ! [delete]create $ANL : private Analysis
<KP_PERIOD>       !         create $ANL : standard Analysis
[<PF1>]<KP_5>     ! [delete]create $DSP : Display
[<PF1>]<KP_6>     ! [delete]create $TMR : Transport Manager
[<PF1>]<KP_COMMA> ! [delete]create $DBM : Data Base Manager
```

Of cause, all GOOSY commands can be entered by line directly to the GOOSY prompter.

The upper key values are the simple key hits, the lower are entered with a preceding GOLD(=PF1)-key hit.



Ctrl Z: Leave GOOSY Prompter

Figure 20.2: The Special Keypad Layout for the GOOSY prompter.

20.1.2 GOOSY Components

Transport Manager

This is the central data dispatcher. It controls the frontend equipment, gets data buffers and dispatches them to DECnet channels, mailboxes and files (disk or tape). Presently it supports CAMAC by the MBD branch driver or by a single crate controller J11. There is a menu of Transport Manager commands available which can be activated by:

SUC: GOOSY> `KP_9`

The menu is self-explanatory and contains short descriptions of the available commands.

Data Base Manager

This component executes all commands to maintain Data Bases and Data Elements, like CREATE, DELETE, SHOW, MODIFY, COPY etc. The menu is activated by:

SUC: GOOSY> `KP_MINUS`

The Data Base Manager may be started directly under DCL. In this case it is called by the DCL command

```
$ MDBM
SUC: DBM>
```

Now the `NEXT SCREEN` key will enter the menu.

Display

This component executes all display commands. It allocates the output device. It gets data from analysis programs via DECnet for scatter plots. The display menu is activated by:

SUC: GOOSY> $\boxed{\texttt{KP\_8}}$

You can start one more display in the same environment by the DCL command:

```
$ GOOSY CREATE PROCESS DISP name
```

where name is a 4 character name. However, if you control two displays from one terminal, you have to prefix all display commands by `name>` ($DSP is the default display):

```
SUC: GOOSY> $DSP> DISPLAY PICTURE
SUC: GOOSY> name> DISPLAY PICTURE
```

The process is started by default with priority 3. Specify another priority by `CREATE PROCESS ... PRIO=p`. The display may be started directly under DCL. In this case it is called by

```
$ MDISP
 SUC: DISP>
```

Now the $\boxed{\texttt{NEXT SCREEN}}$ key will enter the menu. However, no scatter plots can be displayed in this 'stand alone' mode.

Analysis

This user specific program analyzes the data. It may get data buffers from DECnet, Mailbox or files (disk or tape). It may output data buffers to DECnet and files. The $ANL menu is activated by:

SUC: GOOSY> $\boxed{\texttt{KP\_7}}$

Others

Other components may be started at any time. They may execute commands and control Data Elements or hardware components.

As shown above, the Data Base Manager and the Display Program may be executed stand alone on DCL level or bundled together with other GOOSY programs in an environment. The Transport Manager and Analysis components can run ONLY in an environment. Figure 20.1 on page 254 shows the communication between environment components. Commands executed by environment components are dispatched by the GOOSY prompter from one terminal. Therefore, on each terminal one has to create an environment. Commands given from that terminal are executed by the components running in that environment. The Data Bases, however, are shared between environments. Therefore the display may run in a different environment than the analysis.

20.1.3 Creation of Environments

To create an environment with optional components, use the DCL command `CRENVIR`:

```
$ CRENVIR ?  ! Enter menu
$ CRENVIR environment myanal /ONLINE
$ CRENVIR environment myanal /OFFLINE
```

The environment name must be unique within a user group on one VAX node. It can be one to four characters long. The qualifiers create full environments for online or offline. The difference is that an online environment has the TMR component which is not needed offline. Optionally you may specify the name of a private analysis program 'myanal' (default name is MGOOANL). You can use a standard GOOSY analysis program by qualifier `/DEFAULT`:

```
$ CRENVIR environment /ONLINE/DEFAULT
$ CRENVIR environment /OFFLINE/DEFAULT
```

This analysis program executes dynamic lists. An analysis routine can be loaded dynamically. The command `CRENVIR` can be used also for an existing environment to create additional components. The standard components can be created individually by:

```
$ CRENVIR environment /$TMR/$DSP/$ANL/$DBM/J11
                  ! optional /$TMR creates the $TMR component
                  ! optional /$DSP creates the $DSP component
                  ! optional /$DBM creates the $DBM component
                  ! optional /$ANL creates the $ANL component
                                   (user analysis)
                  ! optional /J11  creates the $ANL component
                                   (standard analysis)
```

Any other components can be created by

```
$ CRENVIR environment program name
```

where 'name' may have one to four characters. The analysis started by default with priority 3. Specify another priority by `CRENVIR ...  /PRIO=p`.

20.1.4 Deletion of Environments

The DCL command

```
$ DLENVIR
```

deletes the present environment including all components (subprocesses).

20.1.5 Environment Logfiles

The commands executed in an environment and the command output are logged in three files. The file names are composed by the node name and the environment name:

```
SLOG_node_environment.LOG  ! main log file
CLOG_node_environment.LOG  ! commands only
GLOG_node_environment.LOG  ! create/delete commands only
```

These logfiles are never deleted, especially not by the `PURGE` command, because GOOSY appends the output always to an existing logfile. Therefore one should sometimes have a look at the size of the logfiles and delete them, if they are not used any more. They are created automatically.

Comments in Logfiles

Sometimes it is useful to write comments into the logfile. This can be done by command `PROTOCOL`:

```
GOOSY> PROTOCOL "changed beam from U to PB"
```

Chapter 21

GOOSY Transport Manager

21.1 Introduction

The Transport Manager program (TMR) is a GOOSY component and is created in a GOOSY environment. In terms of VMS it runs in a subprocess. The TMR executes several commands concerning the data acquisition and data dispatch, and the control of CAMAC equipment. Presently it supports the VME frontend system, the MBD and the J11 single crate system. The event data are collected by the frontend processors in formatted data buffers which are input to the TMR and dispatched to several output channels. The TMR supports presently seven types of input channels and three types of output channels. Five of the input channels are exclusive. Only one exclusive input channel can be activated at the same time but several output channels. A more detailed description follows in the next sections.

Input Channels are:

1. VME exclusive
   Data buffers are read from VME-subsystem. The event format is 10,n.
2. MBD exclusive
   Data buffers are read from MBD. The event format depends on the J11 programs.
3. J11 exclusive
   Data buffers are read from J11 via DECnet. A standard buffer and event format is generated by the J11.
4. File exclusive
   Data buffers are read from a disk or tape file (Standard RMS format).
5. Foreign exclusive
   This input channel requires some programming work to support other frontend systems.

TMR: Transport Manager Program
MBD: Microprogrammed Branch driver
J11: Auxiliary crate controller
VME: VME frontend system

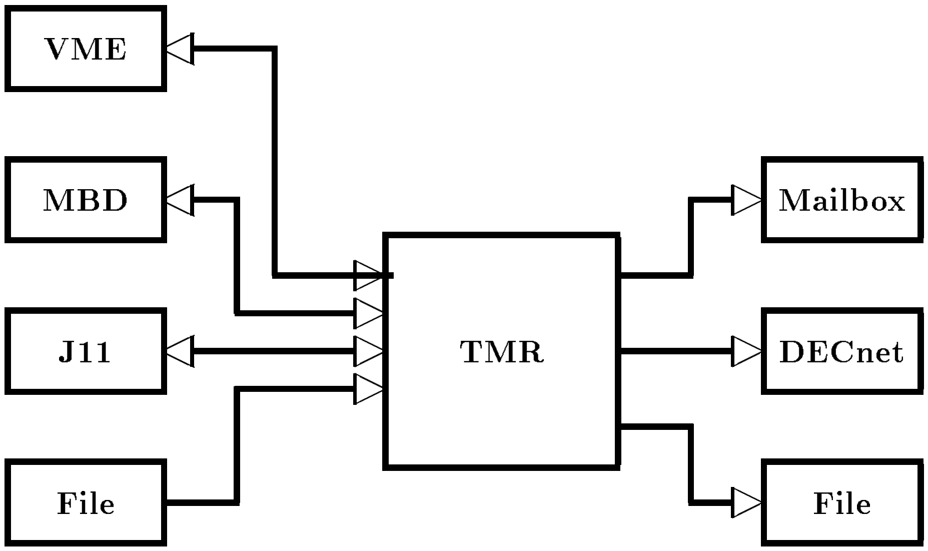


Figure 21.1: The input and output channels of the Transport Manager

6. Mailbox
   At any time a GOOSY buffer may be sent to this channel. It is processed like other buffers.
7. DECnet
   At any time a GOOSY buffer may be sent to this channel. It is processed like other buffers.

The exclusive input channel is selected by the `INITIALIZE ACQUISITION` command. The mailbox and DECnet channels are opened by the sending programs.

Output Channels are:

1. File
   Data buffers are written to a disk or tape file (standard RMS format). This channel

is opened and closed by commands. It always synchronizes the input.

2. Mailbox
   Data buffers are written to mailboxes. There are three mailbox channels. They are filled if they have been read by some program, normally an analysis program. The first mailbox channel optionally synchronizes the input.
3. DECnet
   Data buffers are written to DECnet. Up to 20 DECnet links can be established. The TMR sends buffers to all programs having established a link and having acknowledged the previous buffer. These channels optionally synchronize the input.

Parallel to the data stream dispatching the TMR executes commands controlling the experimental setup, i.e. downline loading programs to frontend processors or executing CAMAC commands. Optionally the data buffer structure is checked and contents may be displayed to the terminal.

21.2 Startup the TMR

The TMR must be started in a GOOSY environment. This can be done by

```
$ CRENVIR environment /$TMR
```

or if the environment exists already :

```
$ GOOSY
GOOSY> CREATE PROCESS TMR $TMR
GOOSY> DELETE PROCESS $TMR
```

The process is started by default with priority 3. Specify another priority by `CREATE PROCESS ... PRIO=p`. The second command deletes the TMR. This does not affect other components of the environment, except that analysis programs cannot get data any more and DECnet links are aborted. It is, however, a good praxis to STOP the acquisition before deleting the TMR component. If the TMR component is created, the TMR must be initialized. At this point one must know about the data input. Therefore the data input is described in the next section.

21.3 Input Channels

21.3.1 MBD Input

Before you can access an MBD, you must specify which MBD you want to use. This is done by the DCL command `SELECT_MBD`. On the cluster VAXes DONALD and EMMA there are two MBD's labeled 'A' and 'B'. The branch cables are labeled as 'DA' and 'DB' or 'EA' and 'EB', respectively.

INITIALIZE ACQUISITION

Using the MBD the TMR must be initialized by

```
GOOSY> INITIALIZE ACQUISITION mailbox size /MBD
```

'Mailbox' is a name used for the creation of the mailboxes. If not specified, the name of the environment is used (this is recommended). The mailbox names are then

```
GOOSY_mailbox_1, GOOSY_mailbox_2, GOOSY_mailbox_3
```

'Size' specifies the buffer size in bytes. The default size is 8192. The minimum size is 512 bytes (for file header). A multiple of 512 should be used. A channel is assigned to device 'MBD', the internal buffer queues are created and the mailboxes are created.

LOAD MBD

For a description of the MBD and J11 hardware and programming see the GOOSY Hardware Manual. The MBD code must be loaded to the MBD, the J11 code to the J11's (one per crate). This is done for a two crate system by

```
GOOSY> LOAD MBD GOO$IO:EXEC/EXEC
GOOSY> LOAD MBD GOO$IO:ESONE
GOOSY> LOAD MBD GOO$IO:C2             ! for 2 crates
GOOSY> LOAD STARBURST file1 1 23/BOOT  ! boot crate one
GOOSY> LOAD STARBURST file2 2 23/BOOT  ! boot crate two
```

The MBD code is in file GOO$IO:EXEC.BDO, the ESONE code in GOO$IO:ESONE.BDO, standard MBD programs for n crates are GOO$IO:Cn. The J11 crate controller programs are loaded by the last command for each crate. For the programming of the J11's see GOOSY Hardware Manual. The CAMAC station number of the J11 is always 23. Now the MBD should be ready to send data.
If the MBD could not be loaded because of an internal loop or any MBD hangup you may try to reset the MBD by the TMR command

```
GOOSY> RESET MBD
```

Be shure what you are doing when loading or resetting an MBD and check that you have selected the right MBD on the right VAX. Otherwise you may destroy running experiments.

21.3.2 VME Input

Before one can proceed with the following commands, the VME-hardware must be set up properly. This is described in the hardware manual. The communication processor (E5 or E6) must run the program `net_slave`. On the VAX some definitions have to be done by DCL command ETHDEF:

```
$ ETHDEF processor device
```

where 'processor' is the name of the E5, i.e. E5ELXA and 'device' is the bus where the ethernet interface is plugged in (QB, UB, BI or WS). This command also defines the two logical names EB_EXEC and FEP_EXEC to the standard FIC software, when they are not yet defined. These names can be used in the setup files (see below).

VME setup files

The next things needed are the text files to describe the VME processors and the readout tables. Examples can be found on directory GOO$EXAMPLES (file types .VMEP and .VMET). The syntax is described with the LOAD VME commands. The equipment controlled by one frontend processor is called a branch. The branch is specified by the VME crate number and the memory offset of the processor indicated by LEDs on the frontpanel. This offset can be set by a switch on the platine. For flexibility an ID number (between 1 and 16k) can be defined by the user for these two numbers. This is done together with the processor specification in the setup file. The ID is written into the subevent header of the branch. Therefore programs may be sensitive to that ID, but the processor offsets may be changed independently. Normally there is one top file specifying only the processors and their ID's. In addition in that file the files specifying the readout should be invoked by @file. This allows to keep the files modular. It is recommended to keep the readout tables for each processor in separate files. The top file can be given to the LOAD VME PROGRAM or TABLE commands described later. The DCL command `VMETREE` displays the whole tree of a top file.

INITIALIZE ACQUISITION

Using the VME frontend the TMR must be initialized by

```
GOOSY> INITIALIZE ACQUISITION mailbox size /VME
```

'Mailbox' is a name used for the creation of the mailboxes. If not specified, the name of the environment is used (this is recommended). 'Size' specifies the buffer size in bytes. The default is 16 kB. The frontend system must respond with "Link open acknowledged". Unlike with the other frontends, the command can be repeated, i.e. if the link to the frontends has been broken.

When the communication processor (E5 or E6) had to be restarted, the transport manager must first close the link by

```
   GOOSY> $ CLOSE ETHERNET
```

When the communication processor does not respond, one has to wait for timeout. Then the `INIT AC /VME` command can be executed.

LOAD VME frontend processors

Now the programs must be loaded to the VME processors. This is done by

```
   GOOSY> LOAD VME PROGRAM @file /TABLE
```

The file must contain a list describing the processors in the VME crate. Default file type is .VMEP. An example can be found on GOO$EXAMPLES:VME_SETUP.VMEP. With the optional qualifier `/TABLE` the readout tables are loaded also (see below). This makes sense only, if the readout files are called in the specified file with the @file option. Example of output (1 CAMAC crate):

```
SUC: GOOSY> LOAD VME PROG @ISETUPC_E6
Load programs from EE$ROOT:[SCHALL.GOOSY]ISETUPC_E6.VMEP;
-> ISETUPC_E6
--> ICAM1.VMET;11
 Crate=1 Processor=4 ID=100 Index=0 Subcrate=0 Control=1 EB EB EB_EXEC
 Crate=1 Processor=5 ID=10 Index=1 Subcrate=0 Control=3 FEP CAV FEP_EXEC
Load system file VME$EXE:EB_ROOT.EX20
Load system file VME$EXE:FEP_ROOT.EX20
 Message from crate 1, offset 4, subcrate 0, control 1.
 > EB GEB 3.0 with 1[512Kb] started, 15 buffers [16 Kb]. FEPs: 1 sync. 0 async.
 Message from crate 1, offset 5, subcrate 0, control 3.
 > FEP GFP 3.0 with 2[512Kb] started. 32 buffers [24026 b] Type=3 Number=1.
```

Once loaded the VME system runs independently of the Transport Manager. When the TMR has to be restarted for some reason, the VME system need not to be loaded again. The TMR must know, however, the VME setup. To achieve that, type

```
SUC: GOOSY> LOAD VME PROG @ISETUPC_E6 /NOLOAD
```

There comes a lot of output which can be ignored.

LOAD VME readout tables

Now the readout tables must be loaded to the VME processors. This is done by

```
   GOOSY> LOAD VME TABLE @file trigger
```

The file is normally the same as above. It contains lines @file referencing file which contain a list describing the modules in the CAMAC and Fastbus crates. A trigger number may be specified between 1 and 15. Default is 1. The lists for different triggers cannot be reloaded separately. If one already loaded list is reloaded, the lists for all other triggers must be reloaded too. Example (1 CAMAC crate):

```
SUC: GOOSY> LOAD VME TABLE @ISETUPC_E6
Load tables from   EE$ROOT:[SCHALL.GOOSY]ISETUPC_E6.VMEP;
-> EE$ROOT:[SCHALL.GOOSY]ISETUPC_E6.VMEP;
--> ICAM1.VMET;11
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Readout tables for all triggers reset!
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  1: Readout for Crate 1, Length 1 loaded.
```

Examples of reaout tables can be found on GOO$EXAMPLES:*.VMET.

SHOW VME SETUP

When the programs and tables are loaded, the command

```
   GOOSY> SHOW VME SETUP
```

displays a list of processors and loaded files. Example:

```
SUC: GOOSY> SHO VME SETUP
----------------------------------------------------------------------
Setup file  : EE$ROOT:[SCHALL.GOOSY]ISETUPC_E6.VMEP
Number FEPs: 1 synchron 0 asynchron
-#---id---cr-pr-sc-cl-mode-cl--ty---------system program----------------
  0   100  1  4  0  1 sync EB  EB  VME$ROOT:[VMEMAN.VMELIB.EXE]EB_ROOT
  1    10  1  5  0  3 sync CAV FEP VME$ROOT:[VMEMAN.VMELIB.EXE]FEP_ROOT
----------------------------------------------------------------------
    Trigger   1 EE$ROOT:[SCHALL.GOOSY]ISETUPC_E6.VMEP;
----------------------------------------------------------------------
SUC: GOOSY>
```

Select Data Transport

There are two ways to get the data from the VME system into the VAX. The faster one is to use a parallel interface (HVR). The second is to use ethernet. The GOOSY event builer must be setup to know which channel to use. This is done by command

```
   GOOSY> SET VME INPUT /NET/HVR/OFF
```

Of cause, only one of the options can be selected. When `/NET` or `/OFF` is selected, you must use the `/NET` qualifier with `START ACQUISITION`. When you select the NET option, the ethernet communication must be set to stream mode (not yet implemented):

```
GOOSY> $ SET GNA ETHERNET /ISTREAM
```

Commands

All other commands given to the VME frontend system have to specify the destination processor(s). This can be done in different ways:

- Specify the processor id with ID=number. This number is defined in your setup files. Only one destination is possible.
- Specify a list of VME-crate/offset pairs of the processors by PROCESSOR=list
- Select processors by type, i.e. /EB or /FEP or /ALL.
- Select processors by type of readout, i.e. /CVI or /CAV or /AEB.

Setup Readout buffers

The frontend processors provide per default 32 subevent buffers of fixed length. The length is calculated from the memory available. The number of buffers or the buffer size may be changed by command

```
GOOSY> SET VME BUFFER BUFFERS=b SIZE=s ...
```

Only one, number of buffers or size [Bytes], can be specified. The other is calculated from the memory available. There is, however, a maximum number of buffers. The number of buffers need not be the same on all processors. The size must be the maximum subevent size. The command answers with a message telling the buffer setup. Note that you must specify the destination in the way described above.

SHOW VME CONTROL

When the programs and tables are loaded, the command

```
GOOSY> SHOW VME CONTROL ...
```

displays buffers, events, and tables loaded by the FEP's. Example:

```
  GOOSY> SHOW VME CONTROL /ALL
Message from crate 1, offset 5, subcrate 0, control 1.
> Bufind(FEP)=0 Events=0 Reject=0 Buffers=0
Message from crate 1, offset 6, subcrate 0, control 3.
> Bufferindex=-1 Events=0 Reject=0 Buffers=0
```

```
 Message from crate 1, offset 6, subcrate 0, control 3.
 > INIT list : Crate 1, Address 20012990, Length 4
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  1: Crate 1, Address 2002A998, Length 1
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  2: Crate 1, Address 2002A99C, Length 2
```

SET Trigger

The trigger modules are set by command

```
   GOOSY> SET VME TRIGGER ID=id /RESET/ENABLE/MASTER FASTCLEAR=f CONVERSION=c
```

Only arguments specified are changed in the trigger module. The module responds with a message displaying status and time settings. A more detailed description of the trigger handling is in the hardware manual. The trigger of FEP 1 must be set to `/MASTER`. All trigger modules must be reset and enabled after power up or hardware problems. Note that you must specify the destination in the way described above. An example for setting up trigger modules can be found in GOO$EXAMPLES:VME_TRIGGER.GCOM. Example:

```
     GOOSY> SET VME TRIG ID=10 /RES/ENAB/MAST FAST=10 CONV=20
 Message from crate 1, offset 6, subcrate 0, control 3.
 > cnt=1 st=00000120 ctrl=00000005 fclr=10 cvt=20 100ns
```

Debug Output

When you have terminals (or windows) connected to your frontend processors, you can enable some output to these windows.

```
   GOOSY> SET VME CONTROL FIC_DEBUG 1 ...
```

A value of 0 disables output. Note that you must specify the destination in the way described above.

Initialize CAMAC

In the readout tables CNAFs can be specified for readout or initialization. The initialization CNAFs are executed by command:

```
   GOOSY> INITIALIZE CAMAC ...
```

Note that you must specify the destination in the way described above.

VME messages

Messages delivered by the frontend processors are written to terminal. A line displaying the source is written first, e.g.

```
Message from crate 1, offset 4, subcrate 0, control 1.
> EB GEB 3.0 with 1[512Kb] started, 15 buffers [16 Kb]. FEPs: 1 sync. 0 async.
Message from crate 1, offset 5, subcrate 0, control 3.
> FEP GFP 3.0 with 2[512Kb] started. 32 buffers [24026 b] Type=3 Number=1.
```

CNAF

There are two ways to execute CNAFs. The first is the standard ESONE mechanism. One has to define the logical name `CAMAC_BRANCH_n` in a similar way as with the MBD or J11 systems. There is one extension to specify the VME branch where to execute the CNAF. The VME destination is specified by `crate:offset`. Offset 30 means the communication processor itself which may have a VSB branch connected.

```
$ DEFINE/JOB CAMAC_BRANCH_0 1:3  ! For CAMAC CNAF command in TMR
$ DEFINE/JOB CAMAC_BRANCH_0 1:30 ! For CAMAC CNAF command in TMR
GOOSY> CAMAC CNAF ...
$ DEFINE/JOB CAMAC_BRANCH_0 node::env____$TMR(GN_ESONE)1:3
```

The last definition is used by programs other than the TMR. The ESONE buffers are routed through the TMR to communication processor to destination processor and back. 'env' is the environment name of the TMR.

The other way is the command

```
GOOSY> CNAF VME ...
```

which works only in the TMR. This command should be used for single CNAFs mainly. It responds with a message. No definitions are necessary.

START/STOP ACQUISITION

When the VME system is set up it is ready for getting data. The `START ACQUISITION` command resets the buffer queues and sets the GO bit in the master trigger module. Note, that you must use the `/NET` qualifier when transfer is switched off or set to `/NET` (SET VME INPUT command). The `STOP ACQUISITION` command clears the GO bit and the master FEP (the first FEP in the setup file which controls the master trigger module) marks a last event. When the event builder encounters the last event, it delivers a message and marks the last buffer. When the TMR encounters a last buffer, it delivers a message similar to the one from the event builder. Up to that time a `START ACQUISITION` command is blocked. If no data is sent to the VAX, e.g. if VME INPUT is switched OFF, the qualifier `/RESET` is needed with the `START ACQUISITION` command.

TYPE EVENT

The command displays event data on the terminal if CHECK mode is active (see also below).

```
   GOOSY>TYPE EVENT ID=id
```

The specification of an ID applies for VME data only. Only subevent data from subevents of the specified branch are displayed. From the other events only the subevent headers are displayed. Specifying a nonexisting ID results in a very compact output.

Example VME session with one CAMAC crate

```
T:GOOFY$A3$ ethdef e6elxb qb
Ethernet device DEV_QB = XQA0:, Protocol TMR = 5380X, Data interface UUA0:
Communication processor E6ELXB with address 00-00-5B-00-03-64
Event builder program EB_EXEC  is VME$EXE:EB_ROOT.EX20
Frontend  program     FEP_EXEC is VME$EXE:FEP_ROOT.EX20
T:GOOFY$A3$ crenv susi/$tmr
HVR connected from MVIIH::SUSI____$TMR(GN_XX_PRCTRL)
GOOSY environment SUSI created
----------------------------------------------------
GOOSY environment process                          Node: MVIIH           Process name: GN_SU
GOOSY component process
GOOSY component process There are 3 processes in this job:
GOOSY component process
GOOSY component process   GN_SUSI (*)
GOOSY component process     GN_SUSI____$SVR
GOOSY component process     GN_SUSI____$TMR
T:GN_SUSI$ GOOSY
SUC: GOOSY> ini ac/vme
Link open acknowleged
SUC: GOOSY> load vme p @test
Load programs from EE$ROOT:[GOOFY.G.VME]TEST.VMEP;
-> TEST
--> TEST11.VMET
 Crate=1 Processor=5 ID=100 Index=0 Subcrate=0 Control=1 EB EB EB_EXEC
 Crate=1 Processor=6 ID=10 Index=1 Subcrate=0 Control=3 FEP CAV FEP_EXEC
Load system file VME$EXE:EB_ROOT.EX20
Load system file VME$EXE:FEP_ROOT.EX20
 Message from crate 1, offset 5, subcrate 0, control 1.
 > EB GEB 3.0 with 2[512Kb] started, 47 buffers [16 Kb]. FEPs: 1 sync. 0 async.
 Message from crate 1, offset 6, subcrate 0, control 3.
 > FEP GFP 3.0 with 1[512Kb] started. 32 buffers [7578 b] Type=3 Number=1.
SUC: GOOSY> loa vme t test1 1
Load tables from   EE$ROOT:[GOOFY.G.VME]TEST1.VMEP;
-> EE$ROOT:[GOOFY.G.VME]TEST1.VMEP;
--> TEST11.VMET
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Readout tables for all triggers reset!
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  1: Readout for Crate 1, Length 1 loaded.
SUC: GOOSY>
```

```
SUC: GOOSY> loa vme t test2 2
Load tables from   EE$ROOT:[GOOFY.G.VME]TEST2.VMEP;
-> EE$ROOT:[GOOFY.G.VME]TEST2.VMEP;
--> TEST12.VMET
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  2: Readout for Crate 1, Length 2 loaded.
SUC: GOOSY> ini cam/cav
SUC: GOOSY> set vme inp/hvr
SUC: GOOSY> sho vme set
--------------------------------------------------------------------------------
Setup file  : EE$ROOT:[GOOFY.G.VME]TEST.
Number FEPs: 1 synchron 0 asynchron
-#---id---cr-pr-sc-cl-mode-cl--ty---------system program------------------
  0   100  1  5  0  1 sync EB   EB  VME$ROOT:[VMEMAN.VMELIB.EXE]EB_ROOT
  1    10  1  6  0  3 sync CAV FEP VME$ROOT:[VMEMAN.VMELIB.EXE]FEP_ROOT
--------------------------------------------------------------------------------
    Trigger   1 EE$ROOT:[GOOFY.G.VME]TEST1.VMEP;
    Trigger   2 EE$ROOT:[GOOFY.G.VME]TEST2.VMEP;
--------------------------------------------------------------------------------
SUC: GOOSY> sho vme con /all
 Message from crate 1, offset 5, subcrate 0, control 1.
 > Bufind(FEP)=0 Events=0 Reject=0 Buffers=0
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Bufferindex=-1 Events=0 Reject=0 Buffers=0
 Message from crate 1, offset 6, subcrate 0, control 3.
 > INIT list : Crate 1, Address 20012990, Length 4
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  1: Crate 1, Address 2002A998, Length 1
 Message from crate 1, offset 6, subcrate 0, control 3.
 > Trigger  2: Crate 1, Address 2002A99C, Length 2
SUC: GOOSY>
SUC: GOOSY> set vme trig id=10 /res/enab/mast fast=10 conv=20
 Message from crate 1, offset 6, subcrate 0, control 3.
 > cnt=1 st=00000120 ctrl=00000005 fclr=10 cvt=20 100ns
SUC: GOOSY> sta ac
SUC: GOOSY> ty ev

********************* Buffer ***************************************************
Buffer type =   10,      Buffer number =        6,      Total length =     8168
Subtype =        1,      Events  =            510,      Data  length =     8160
No spanning events!                                     29-JUL-1992 18:06:47.81
********************************************************************************
```

```
==================== Event 1    ==============================================
Type =   10, Subtype =    1, Length =     12, Trigger =  1, Event =        2551
Type =   10, Subtype =    1, Length =      4, Crate =    1, ID =    10, Ctrl CAV
1       :32768             :                  :                 :

SUC: GOOSY>
SUC: GOOSY> stop ac
GN_SUSI____$TMR: ===>  Acquisition stopped
 Message from crate 1, offset 5, subcrate 0, control 1.
 Event builder sent last buffer
 > EB:  Events=7210, Last buffer=15, evts=70, words=1120, t=0, e=
   TMR: Events=7210, Last buffer=15
GN_SUSI____$TMR: STOP ACQUISITION finished.
SUC: GOOSY> $ cl eth
Link aborted to Frontend.
SUC: GOOSY> $ EXIT
T:GN_SUSI$ dlenv
Subprocess deleted: GN_SUSI____$TMR
Subprocess GN_SUSI____$TMR has completed
Subprocess GN_SUSI____$SVR has completed
```

21.3.3 J11 Input

INITIALIZE ACQUISITION

Using the J11 the TMR must be initialized by

```
   GOOSY> INITIALIZE ACQUISITION mailbox size /J11 NODE=j11_node
```

'Mailbox' is a name used for the creation of the mailboxes. If not specified, the name of the environment is used (this is recommended). 'Size' specifies the buffer size in bytes. The default is 8 kB. 'J11_node' is the DECnet node name of the J11. Two links are opened to the specified DECnet node, one for commands, one for data, the internal buffer queues are created and the mailboxes are created.

LOAD J11

Now the CAMAC setup file must be loaded to the J11. This is done by

```
   GOOSY> LOAD J11 file
```

The file must contain a list describing the modules in the crate. For details see GOOSY Hardware Manual. Now the J11 should be ready to send data. The J11 may be initialized several times and can be loaded several times. The acquisition must be stopped to do that.

21.3.4 File Input

INITIALIZE ACQUISITION

Using the file input the TMR must be initialized by

```
GOOSY> INITIALIZE ACQUISITION mailbox size /FILE
```

'Mailbox' is a name used for the creation of the mailboxes. If not specified, the name of the environment is used (this is recommended). 'Size' specifies the buffer size in bytes. The default is 8 kB. The buffer size equals the file record size. The internal buffer queues and the mailboxes are created.

OPEN-CLOSE FILE

Now the input file must be opened. This is done by

```
GOOSY> OPEN FILE file
```

Now the TMR is ready to read data from the file.

21.3.5 Foreign Input

INITIALIZE ACQUISITION

This input channel can be used to support other frontends. The routines to control the frontend must be provided by the user.

21.3.6 START-STOP ACQUISITION

When the input channel is ready the data taking is started by

```
GOOSY> START ACQUISITION buffers events skip_buf skip_event
```

The optional parameters specify the number of buffers or events to process and the number of buffers or events to skip first. A number of zero means unlimited. This is the default.

```
GOOSY> STOP ACQUISITION
```

stops the data taking. All data in the frontends are read and written to the output file, if a file is open and started. A J11 system can be stopped in the way that all data buffers in the J11 are aborted by:

```
GOOSY> STOP ACQUISITION /ABORT
```

The links terminate. You must INIT the ACQUISITION again to start after a /ABORT. If the data input comes from a file, this file can be closed by

```
GOOSY> STOP ACQUISITION /CLOSE
```

To start the acquisition again (reading from a file) a file must be opened.
NOTE: The output file is not closed by STOP ACQUIS.

START-STOP Routines

It is possible to load private routines into the Transport Manager which are then called during START and STOP ACQUIS. See the `LOAD MODULE ACQUISITION` command in section 21.6 for that. These routines must be linked into a sharable image by the DCL command `LSHARIM` (see section 22.5.1 page 296.

21.4 CAMAC Spectra

CAMAC spectra can be incremented in a MR2000 CAMAC memory. The spectra must be created as normal GOOSY spectra with additional information about the crate, station and offset of the spectrum in the CAMAC memory. Commands are provided to copy or add the contents from CAMAC memory into the GOOSY spectrum or to copy or add the contents of the GOOSY spectrum into the CAMAC memory. The accumulation of the CAMAC spectra is controlled by special `START` and `STOP` commands. Note that these commands are executed in the Data Base Manager which must therefore be running.

```
GOOSY> START MR2000 branch crate station /INIT
GOOSY> STOP  MR2000 branch crate station
GOOSY> CLEAR CAMAC SPECTRUM name /CAMAC/SPECTRUM
GOOSY> READ  CAMAC SPECTRUM name /ADD
GOOSY> WRITE CAMAC SPECTRUM name /ADD
GOOSY> SHOW  CAMAC SPECTRUM name       !contents of MR2000
```

21.5 Output Channels

The data buffers collected from the input channel are dispatched to one or more output channels. Normally analysis programs connect to these channels to get the data buffers. The TMR input may be synchronized by an analysis program. This mode must be enabled by a command (see `SET ACQUISITION` command). By default the output channels are filled only with samples, that means if the receiver has acknowledged a buffer. The file output channel synchronizes the input always.

21.5.1 Mailbox Output

Three mailboxes are scanned for output. One buffer is written to each mailbox after starting the acquisition. The second buffer is written to those mailboxes read by an analysis program. If a mailbox has not been read, no more buffers are written to this mailbox until it was read. The first mailbox can be used to synchronize input. This mode is enabled by

```
GOOSY> SET ACQUISITION /SYNC/MAIL
```

Then the TMR waits for the readout of this mailbox before initiating a new input. **NOTE: Mailbox input should be started first before setting synchronous mode**. An analysis program starts mailbox input by

```
GOOSY> START INPUT MAIL mailbox number
```

where 'mailbox' must be the same name as specified in the `INITIALIZE ACQUISITION` command and 'number' is 1,2 or 3.

21.5.2 DECnet Output

DECnet output channels are opened for analysis programs by

```
GOOSY> START INPUT NET node environment
```

where 'node' is the VAX node of the TMR and 'environment' is the name of the environment of the TMR. All opened DECnet channels are scanned for output. One buffer is written to each channel after it has been opened. The second buffer is written to those channels which received an acknowledge from the connected analysis program. DECnet channels can be used to synchronize input. This mode is enabled by

```
GOOSY> SET ACQUISITION /SYNC/NET
```

Then the TMR waits until the buffer is acknowledged by any of the connected analysis programs before initiating a new input. **NOTE: DECnet input should be started first before setting synchronous mode**. The same buffer may be written to several channels unless exclusive mode is enabled by

```
GOOSY> SET ACQUISITION /EXCLUSIVE
```

Then a buffer is sent only to one (free) channel. If only one channel is open, all buffers input to the TMR are analyzed. If several channels are open, all buffers are analyzed but by several analysis programs. NO buffer is sent to more than one analysis.

21.5.3 File Output

This channel synchronizes the TMR input. It must be explicitly started and stopped by commands. After opening an output file, file writing can be started and stopped without closing the file. The file can be on disk or on tape. A tape to be used must be mounted (see tape handling on page 281). This output channel is filled regardless of analysis programs.

START-STOP OUTPUT FILE

To start file output to a new file and close a file one types

```
GOOSY> START OUTPUT FILE file size number /OPEN/AUTOMATIC
GOOSY> STOP OUTPUT FILE /CLOSE
```

Then a new file is opened. 'Size' specifies the intended size of the file. When the file is filled it is automatically closed and the acquisition is stopped. All data from the frontends are transferred to the file. Then the file is closed. 'Number' is optionally used together with the `/AUTO` switch. It means that 'number' files of size 'size' are automatically opened, filled and closed. A running number is added to the file name in this case. The `/OPEN` and `/CLOSE` qualifiers are default. To stop file writing and to start writing the same file one types

```
GOOSY> STOP OUTPUT FILE /NOCLOSE
GOOSY> START OUTPUT FILE /NOOPEN
```

The STOP OUTPUT FILE commands does not stop the acquisition.

GOOSY File Header

A GOOSY file header is written to each file after it had been opened. The file header can be copied to/from a disk text file, it can be modified using text editors or by prompting.

```
GOOSY> START OUTPUT FILE HEADEROUT=file /PROMPT
```

The information for the file header is prompted. The optional HEADEROUT specifies a text file to which the header is copied.

```
GOOSY> START OUTPUT FILE HEADERIN=file /EDIT
```

Once a header has been written to a file, it can be used again. The /EDIT qualifier calls first the editor to modify the file. You may also use the /PROMPT qualifier here.

Naming Conventions for IBM

If one wants to send the output files to the IBM, the filenames must follow some conventions:

1. Maximal length 25 char (including type)
2. Maximal 8 char or 7 digits between two underscores (No $).
3. File type must be .LMD

Tape Handling

Writing to tape requires some additional operations. If the tape is new, it must first be initialized and then mounted. The initialization and the mount should be done within the TMR.

```
   GOOSY> MOUNT TAPE tape-device tapename /INIT
```

With these commands the tape density and the size of the tape records can be specified. The defaults should be adequate. The name of the tape is used for the (optional) initialization. After that the tape file will be opened and started like a disk file. You may specify the device together with the file name or as a separate parameter. In the last case the device will be defaulted for following commands.
Use STOP ACQUISITION before STOP OUTPUT FILE and START OUTPUT FILE before START ACQUISITION to make sure that all data sent from CAMAC are written to file!
If a file size limit is specified, the acquisition is stopped automatically early enough to write all buffers to the file.
To dismount the tape issue the GOOSY command:

```
  GOOSY> STOP OUTPUT FILE /CLOSE
  GOOSY> DISMOUNT TAPE device:
```

End of Tape

When the tape runs out of space, a STOP ACQUISITION is executed and the file is closed. All data from the frontends are transferred to the file! However, when the tape was not empty at the beginning,the TMR cannot know the space available on the tape. In this case it may happen, that the tape end is reached. Then VMS rewinds the tape and requires a continuation tape. Mount the next tape on the device and look to the VAX operator console for the number of your tape request. Then type on your terminal in DCL:

```
     $ NEXTTAPE number
```

21.6 Loading Private Routines

After the startup of the Transport Manager one can load private routines to be called by the `START-STOP ACQUISITION` commands.

21.6.1 LOAD MODULE ACQUISITION

These modules must be loaded by the command

```
  GOOSY>LOAD MODULE ACQUISITION image module init /START
  GOOSY>LOAD MODULE ACQUISITION image module init /STOP
```

The optional init parameter is the name of an initialization entry. This entry is called immediately by the command. All these modules must be linked in a sharable image. This is done very convenient by the DCL command `LSHARIM` (see section 22.5.1 on page 296). When the modules are loaded, they are called by the `START-STOP ACQUISITION` commands. Their calling can be switched OFF and ON by the `SET ACQUISITION` command.

21.6.2 Enable/Disable Calling

Enable/disable the calling of loaded start or stop modules is done by

```
   GOOSY> SET ACQUISITION /START
   GOOSY> SET ACQUISITION /NOSTART
   GOOSY> SET ACQUISITION /STOP
   GOOSY> SET ACQUISITION /NOSTOP
```

21.7 Acquisition Synchronization

After startup the TMR only writes samples into the mailbox and DECnet channels. If a file channel is started, this channel synchronizes the input. Sometimes it is necessary to analyze 100% of the incoming data. Then the TMR must be synchronized with the analysis. Enable/disable mailbox synchronization is done by

```
GOOSY> SET ACQUISITION /SYNC/MAIL
GOOSY> SET ACQUISITION /NOSYNC
```

If mailbox synchronization is enabled, the TMR waits for the readout of the first mailbox. Then it fills the open DECnet channels if they are acknowledged and the output file. Enable/disable DECnet synchronization is done by

```
GOOSY> SET ACQUISITION /SYNC/NET
GOOSY> SET ACQUISITION /NOSYNC
```

If net synchronization is enabled, the TMR waits for any acknowledge of a DECnet channel. Then it fills the mailboxes if they have been read and the output file. Enable/disable exclusive output mode is done by

```
GOOSY> SET ACQUISITION /EXCLUSIVE
GOOSY> SET ACQUISITION /NOEXCLUSIVE
```

If exclusive mode is enabled the TMR writes one buffer only in one DECnet channel. Together with `/SYNC/MAIL` this means that only mailbox 1 is filled. Together with `/SYNC/NET` it means that all buffers are analyzed, each by one analysis program.

21.8 Miscellaneous Commands

Besides the commands described above the following commands are available.

21.8.1 Data Checking

After startup the TMR checks the buffer structure of each input buffer. This checking may be disabled. Enable/disable data checking is done by

```
   GOOSY> SET ACQUISITION /CHECK       ! default
   GOOSY> SET ACQUISITION /NOCHECK
```

If checking is enabled the TMR analyzes the data buffers and counts the events. If checking is disabled, the `TYPE EVENT-BUFFER` commands cannot work.

21.8.2 Compress Mode

Selecting J11 compress mode is done by

```
   GOOSY> SET ACQUISITION /COMPRESS
   GOOSY> SET ACQUISITION /NOCOMPRESS  ! default
```

For J11 input these commands control the data format written by the J11. If compression is enabled, the J11 writes no zeros, but the module number followed by nonzero value. If disabled, the J11 writes fixed length events. In both cases the appropriate unpack routine is provided and selected by the Analysis Manager. The events in the Data Base are the same.

21.8.3 SHOW ACQUISITION

This command gives an overview over the TMR activities and modes. A typical output looks like

```
GOOSY> SHOW ACQUISITION

 ------  Status of Data Acquisition: -----------  8-MAR-1988 17:11:47.23
Buffer size   :  8192    Count: 12
Queues: File:     4  LMD:  0  Current: FREE:   12  File:   0  LMD:   0
File input:  CLOSED
Acquisition : STOPPED,  List mode dump: STOPPED,   File: CLOSED
Buffer- and Event-Statistic:
    FILE reads:          0 buffers         0 events since clear
                         0 buffers         0 events since start
    LMD write:           0 buffers since clear
                         0 buffers since start
                         0 buffers since open
```

```
----------------------------------------------------------------------
  Online Analysis statistic:
  GOOSY_SUSI_1 buffers:          0 100%,           0 100%  since clear
  GOOSY_SUSI_2 buffers:          0 100%,           0 100%  since clear
  GOOSY_SUSI_3 buffers:          0 100%,           0 100%  since clear
Enabled:  buffer check -
Disabled: MBX synchronization - NET synchronization - exclusive -
----------------------------------------------------------------------

GOOSY>
```

The output may be directed to a file and optionally printed by

```
   GOOSY>SHOW ACQUISITION file /PRINT
```

The buffer and event counters can be cleared by

```
   GOOSY>SHOW ACQUISITION /CLEAR
```

Only brief information is displayed by

```
   GOOSY>SHOW ACQUISITION /BRIEF
```

The current file headers are displayed by

```
   GOOSY>SHOW ACQUISITION /INFILE/OUTFILE
```

On a separate terminal one can display a continuously updated overview by DCL command

```
    $ GSTAT env /$TMR
```

21.8.4 TYPE EVENT-BUFFER

These commands display buffer or event data on the terminal if CHECK mode is active.

```
   GOOSY>TYPE BUFFER number
   GOOSY>TYPE EVENT number
   GOOSY>TYPE EVENT number /SAMPLE
   GOOSY>TYPE EVENT number ID=id
```

'Number' specifies the number of buffers or events to be typed. The `/SAMPLE` switch works for VME, J11 and MBD buffers. It means that one event per buffer is typed. The specification of an ID applies for VME data only. Only subevent data from subevents of the specified branch are displayed. From the other events only the subevent headers are displayed. Specifying a nonexisting ID results in a very compact output.

21.9 Controlling the Acquisition

When the GOOSY environment and the TMR is created, the following strategy should be used to check if everything works as expected.

21.9.1 Checking Incoming Data

First start the acquisition. Then control if buffers are delivered to GOOSY. Use the `SHOW` command several times, especially if the data rates are low. Inspect the events written by the J11's.

```
GOOSY> START AC   ! start CAMAC readout
GOOSY> SHO ACQ    ! show if data buffers are read
GOOSY> TYPE EV 10 ! inspect 10 events
```

21.9.2 Analyze Data

The next step would be to send data to the analysis to see scatterplots and spectra. It is recommended to use a simple analysis first. You may create a dynamic list doing simple accumulations. This list can be deactivated later. Disable the analysis routine.

```
GOOSY> SET ANAL/NOANAL   ! Disable analysis routine
GOOSY> START INP MAIL    ! Start data input TMR to ANL
GOOSY> SHO ANAL/BR       ! Look if buffers are read
GOOSY> ATT DYN LIST accu ! Activate dynamic list, e.g. accu
GOOSY> ATT DYN LIST $scatter! Activate dynamic list for scatter
GOOSY> SHO ANAL/BR       ! Look if buffers are read
```

Before you attach the dynamic list you may check that data buffers are read into the analysis. Then inspect your data by looking to scatterplots and single spectra as accumulated in the dynamic list.

21.9.3 Modifying Hardware Setup

When you have to adjust the electronic setup, stop the acquisition first, make the electronic ready, clear the spectra, display a new scatterplot and start again.

```
GOOSY> STOP ACQUIS   ! stop CAMAC readout
! Do the modifications in the electronics
GOOSY> CLEAR SPEC *  ! clear spectra
GOOSY> DISP PI       ! display new scatterplot
GOOSY> START ACQ     ! Start CAMAC readout again
GOOSY> TYPE EVENT    ! Inspect incoming data
```

21.9.4 Writing to Tape

When the data looks OK, you may want to start writing to tape. One should always stop the acquisition before starting or stopping the output. Similar, one should always start the output first and then start the acquisition.

```
GOOSY> STOP AC                     ! Stop to reset hardware scalers
GOOSY> MOUNT TAPE M3: label /INIT  ! Mount and initialize the tape
GOOSY> START OUT FILE M3:filename  ! start data writing to tape
GOOSY> START ACQUIS                ! Start CAMAC readout
GOOSY> SHO ACQUIS                  ! Check that buffers are written
GOOSY> STOP ACQ                    ! Stop CAMAC readout
GOOSY> STOP OUT FILE               ! Stop and close output file
GOOSY> START OUT FILE M3:filename  ! start data writing to next file
GOOSY> START ACQUIS                ! Start CAMAC readout
GOOSY> SHO ACQUIS                  ! Check that buffers are written
```

21.9.5 Full Analysis

The data acquisition and output is not affected by the analysis program. To enable the full analysis, one may detach the dynamic list, enable the private analysis routine, clear the spectra. The analysis input may be stopped first.

```
GOOSY> STOP INP MAIL       ! optional stop input
GOOSY> DET DYN LI accu     ! Disable dynamic list. e.g. accu
GOOSY> SET ANAL/ANAL       ! Enable calling of analysis routine
GOOSY> CLEAR SP *          ! Clear spectra
GOOSY> START INP MAIL      ! Start data input
GOOSY> SHO AN              ! Check that data buffers are read
GOOSY> DISP PIC xx         !
```

Chapter 22

GOOSY Analysis Manager

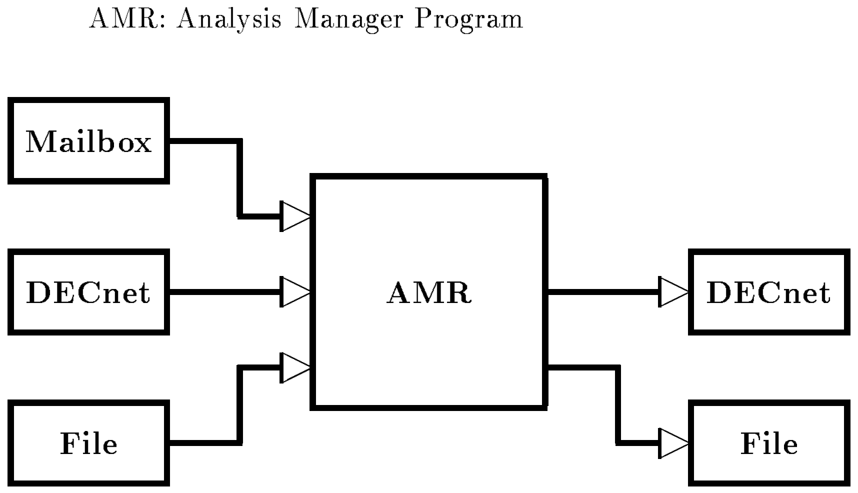


Figure 22.1: The input and output channels of the Analysis Manager (analysis program)

22.1 Introduction

The Analysis Manager (AMR) is part of the analysis program. This program must be created as component in a GOOSY environment. In terms of VMS it runs in a subprocess. The AMR executes several commands concerning the data input/output and the analysis. It supports presently three types of input channels and two types of output channels.

Input Channels are:

1. Mailbox
   Data buffers are read from a mailbox.
2. DECnet
   Data buffers are read from one or several DECnet channels.
3. File
   Data buffers are read from a disk or tape file.

The input channels are selected by the `START INPUT` command.

Output Channels are:

1. File
   The output buffers are written to disk or tape file. This channel always synchronizes the input.

2. DECnet
   The output buffers are written into a DECnet channel. In contrast to the TMR only one channel is available. Optionally this channel synchronizes the input.

Between input and output the AMR executes the event analysis loop. For each event the following steps are executed:

1. Copy and unpack event from buffer into a Data Element in the Data Base.
2. Call user analysis routine.
3. Call dynamic list executor.
4. Copy and pack a Data Element from the Data Base into output buffer. This Data Element is assumed to be filled by the user analysis routine.

All steps except the first one are optional.

22.2 Startup the AMR

The analysis program must be started in a GOOSY environment. This can be done by

```
$ CRENVIR environment /$ANL
```

or if the environment already exists:

```
$ GOOSY
GOOSY> CREATE PROCESS analysis $ANL
GOOSY> DELETE PROCESS $ANL
```

The analysis is started by default with priority 3. Specify another priority by `CRENVIR ... /PRIO=p` or `CREATE PROCESS ... PRIO=p`, respectively. 'Analysis' is the analysis program linked by the user. A standard program is provided on GOO$EXE:MGOOANL. It can handle data buffers from a single crate J11 system or standard MBD and executes dynamic lists. The second command deletes the analysis component. This does not affect other components of the environment. It is a good praxis to STOP the analysis before deleting the analysis component. If the analysis component is created, it is automatically initialized.

22.2.1 Analysis Initialization

During the startup the Data Base DB is attached and all known initialization entries are called. The defaults used are:

Data Base The Data Base name is **DB** which may be a logical name.

Unpack The unpack routine is **X$EVENT** for MBD user buffers, **X$UPMBD** for MBD standard buffers, **X$UPEVT** for J11 buffers, **X$UPCMP** for compressed data buffers and **X$UPEVT** for uncompressed standard data buffers. The routine X$EVENT must be provided by the user, the others are provided by GOOSY. The unpack routine is selected by the buffer type.

Pack The pack routine if needed is **X$PAEVT** for uncompressed event output and **X$PACMP** for compressed event output. The pack routine is selected by the `START ANALYSIS OUTPUT` command.

Input event The input event is written into Data Element **DB:[DATA]EVENT**. For J11 events this Data Element must have type SA$EVENT, for standard MBD events type SA$MBD. The Data Element must exist in the Data Base before creating the analysis component.

Output event The output event written to the output buffers is **DB:[DATA]NEWEVENT**. The Data Element must exist in the Data Base before creating the analysis component. It must have a standard event header (4 words) for uncompressed output mode. The structure is free for compressed output mode.

22.3 Input Channels

Only one type of input channel can be active at a time. However, several DECnet input channels can be opened. This is used if several analysis programs running on different nodes do the calculation part of the analysis and write result events into DECnet channels. Then one master analysis collects the buffers from all these channels and does the histogramming. Depending on the buffer type the proper unpack routine is called.

22.3.1 File Input

The normal OFFLINE analysis reads the data from a disk or tape file.

START-STOP INPUT FILE

The analysis from file is started and stopped by

```
GOOSY> START INPUT FILE file /OPEN buffers events skip_buf skip_events
GOOSY> STOP INPUT FILE /CLOSE
```

These commands open and close the file and start and stop data input. The `/OPEN` qualifier forces an already opened file to be closed first. Optionally the number of buffers or events to processed and to be skipped can be specified. To stop the data input without closing the file and start data input from the same file one uses

```
GOOSY> STOP INPUT FILE
GOOSY> START INPUT FILE
```

If there was no file open, it is opened in any case (the file name is required then).

22.3.2 Mailbox Input

An ONLINE analysis running on the same node as the TMR can get data buffers through one of three mailboxes filled by the TMR.

START-STOP INPUT MAILBOX

To start and stop data input from a mailbox one uses

```
GOOSY> START INPUT MAILBOX mailbox number SIZE=size
GOOSY> STOP INPUT MAILBOX
```

'Mailbox' is a name used for the creation of the mailbox. If not specified, the name of the environment is used. This name must be the same as specified in the `INITIALIZE ACQUISITION` command. If the analysis runs in the same environment as the TMR, the mailbox name should be omitted in both commands. 'Number' is 1,2 or 3. If the TMR runs synchronously to mailbox it waits for number '1'. The mailbox name is then

```
   GOOSY_mailbox_number
```

'Size' specifies the buffer size in bytes. The default is 8192 bytes.

22.3.3 DECnet Input

If an analysis program runs on a different node than the TMR it can get data buffers only via DECnet. It may also get data buffers from other analysis programs. Of cause DECnet channels may also be used at the same node.

START-STOP INPUT NET

Data input from a TMR DECnet channel is started and stopped by

```
   GOOSY> START INPUT NET node environment BUFFERS=buffers EVENTS=events
   GOOSY> STOP INPUT NET
```

'Node' is the TMR node, 'environment' the name of the TMR environment. The number of buffers or events to be processed can be optionally specified. Data input from an analysis DECnet channel is started and stopped by

```
   GOOSY> START INPUT NET node environment component /ANL/MULTI
   GOOSY> STOP INPUT NET
```

'Node' is the node of the other analysis, 'environment' the name of the environment where the other analysis runs. The 'component' parameter specifies the name of the other analysis (normally $ANL). The switch `/MULTI` allows to start DECnet input from several analysis programs. This is used if several analysis programs running on different nodes do the calculation part of the analysis and write result events into DECnet channels. Then one master analysis collects the buffers from all these channels and does the histogramming. The number of buffers or events to be processed can be optionally specified.

22.4 Output Channels

If the analysis program calculates new events it is often useful to generate a new data set. If the analysis is very CPU intensive, it may be necessary to split it into parts, e.g. one analysis reads the data from a file, does the calculations and data suppressions, and sends output to another one which does the histogramming. Then the two programs may run on different nodes. One step further, a TMR may read data from a file, send it to several analysis programs running on different nodes. These programs do all the calculations and send output buffers to one master analysis which does the final analysis.

22.4.1 DECnet Output

It is assumed that the user written analysis routine writes a new event into a Data Element. This Data Element is copied into the output buffer. If the buffer is filled and the channel is opened by another analysis it is written to the channel. Optionally the input may be synchronized by the output.

START-STOP ANALYSIS OUTPUT

The DECnet output is started and stopped by

```
GOOSY> START ANALYSIS OUTPUT /SYNC
GOOSY> STOP ANALYSIS OUTPUT
```

Optionally a type and subtype number can be specified. These numbers are written into each output buffer header. The standard pack routines X$PAEVT and X$PACMP use their own numbers which cannot be changed.

22.4.2 File Output

In addition to the DECnet output the output buffers can be written to a disk or tape file.

START-STOP ANALYSIS OUTPUT

This is started and stopped by

```
GOOSY> START ANALYSIS OUTPUT file size /OPEN
GOOSY> STOP ANALYSIS OUTPUT /CLOSE
```

The file output always synchronizes the input.

```
GOOSY> START ANALYSIS OUTPUT /COMPRESS
GOOSY> START ANALYSIS OUTPUT /COPY     ! default
```

selects between compress pack mode and copy pack mode.

22.5 Loading Private Routines

It is possible to load routines after the startup of the analysis. These routines are called for defined purposes:

- By START INPUT commands
- By STOP INPUT commands
- For event/buffer unpack
- For analysis
- For event pack (output)

The names of these routines can be chosen freely. They may have initialization entries called once. Their argument lists are well defined. They must be linked in a sharable image by DCL command `LSHARIM` (see below). Without loading private modules standard unpack routines are selected by the buffer/event type of the incoming data. The analysis routine X$ANAL and standard event pack routines are called. By loading private modules these modules are called instead of the standard ones. Therefore loaded unpack modules should check if the buffer and event types are correct.

22.5.1 Linking a Private Sharable Image

The command `LSHARIM` should be used to link private modules into a sharable image. The modules must be compiled. They may be in an object library. For a full description use DCL command `HELP LSHARIM`.

```
    LSHARIM module image /GLOBAL=list
         /SHARELOG=name /MAP /KEEP /GROUP
 e.g.:
    $ LSHAR X$START,X$STOP MYSHARE /GR
```

X$START, X$STOP are the file names of the object files. A logical name must be defined for the sharable image file. This is done automatically by the command `LSHARIM`. The logical name is entered in the JOB table. If you want to enter it into the GROUP table (you need the privilege GRPNAM for that), use the /GROUP switch. The logical name is defaulted to the name of the image file.

When the sharable image is ready the GOOSY commands `LOAD MODULE ANAL` and `INIT ANAL` load and activate the routines.

22.5.2 Loading Modules

The analysis must be stopped to load modules. After a module is loaded, the calling of the module is disabled. It must be enabled by the `INIT ANAL` command. The `LOAD MODULE ANALYSIS` command loads one module and optionally its initialization entry.

```
GOOSY>LOAD MODULE ANALYSIS image module init /UNPACK
GOOSY>LOAD MODULE ANALYSIS image module init /PACK
GOOSY>LOAD MODULE ANALYSIS image module init /ANAL
GOOSY>LOAD MODULE ANALYSIS image module init /START
GOOSY>LOAD MODULE ANALYSIS image module init /STOP
```

Example:

```
GOOSY> STOP INPUT ...
GOOSY> LOAD MOD ANAL X$ANAL $XANAL /ANAL
GOOSY> LOAD MOD ANAL X$START $XSTART /START
GOOSY> LOAD MOD ANAL X$STOP /STOP
```

In this example the module X$STOP has no initialization entry.

22.5.3 Enable/Disable Calling of Loaded Modules

When the modules are loaded, their calling must be enabled. Note that the analysis must be stopped and the dynamic lists must be detached. By the command `INITIALIZE ANALYSIS` the initialization entries of all activated modules are called. The NO switches disable calling of a loaded module.

```
GOOSY>INIT ANAL /[NO]ANAL/[NO]START/[NO]STOP/[NO]PACK/[NO]UNPACK
```

Example:

```
GOOSY> STOP INPUT ...
GOOSY> DETACH DYN LIST xxx
GOOSY> INIT ANAL/ANAL/START/STOP
GOOSY> AT DYN LI xxx
GOOSY> START INPUT ...
```

During this command the initialization entries specified in the load commands are called. The calling of the modules is enabled. The calling of the START/STOP modules can be disabled/enabled during a running analysis by

```
GOOSY> SET ANAL/[NO]STOP
GOOSY> SET ANAL/[NO]START
```

These commands do NOT call the initialization entries! The modules for unpack, analysis and pack can be disabled by

```
GOOSY> STOP INPUT ...
GOOSY> DETACH DYN LIST xxx
GOOSY> INIT ANAL/NOANAL/NOUNPACK/NOPACK
GOOSY> AT DYN LI xxx
GOOSY> START INPUT ...
```

22.6 Re-Initialize Analysis

22.6.1 ATTACH-DETACH ANALYSIS

The analysis program attaches to the Data Base. All Data Elements used are locked and cannot be deleted. Sometimes it is necessary to free the Data Base. This is done by

```
GOOSY> DETACH ANALYSIS
GOOSY> ATTACH ANALYSIS
```

The second command does the default initialization as during startup. If modules are loaded, their initialization entries are called.

22.6.2 Setting the Event Data Element

The Data Element used to copy an event from the input buffer to the Data Base is DB:[DATA]EVENT. If another Data Element should be used this can be specified by

```
GOOSY> SET EVENT INPUT DB:[directory]element
```

Similar, the Data Element copied into the output buffers is DB:[DATA]NEWEVENT. If another Data Element should be used this can be specified by

```
GOOSY> SET EVENT OUTPUT DB:[directory]element
```

NOTE that the Data Base DB name cannot be changed by these commands, it can be, however a logical name. With these commands the specified Data Elements are located and all initialization entries of unpack and pack routines are called. The pointer to the Data Element and the size in bytes are passed as arguments.

22.7 Miscellaneous Commands

22.7.1 Enable/Disable Calling of Analysis Routine

The execution of the user analysis routine can be enabled and disabled by

```
GOOSY> SET ANALYSIS /ANALYSIS
GOOSY> SET ANALYSIS /NOANALYSIS
```

22.7.2 Enable/Disable Dynamic List Execution

The execution of the dynamic list executor can be enabled and disabled by

```
GOOSY> SET ANALYSIS /DYNAMIC
GOOSY> SET ANALYSIS /NODYNAMIC
```

22.7.3 Synchronizing the Output

The analysis output can be synchronized by

```
GOOSY> SET ANALYSIS /SYNCHRON
GOOSY> SET ANALYSIS /NOSYNCHRON
```

22.7.4 Select Buffer or Event Unpacking

The calling of the unpack routines can be changed. Per default the pointer to the data buffer is passed as argument to the unpack routines. One can switch to a mode where the unpack routines get the pointer to an event in the buffer:

```
GOOSY> SET ANALYSIS /EVENT
GOOSY> SET ANALYSIS /NOEVENT
```

22.7.5 SHOW ANALYSIS

To get on overview about the analysis activity one types

```
GOOSY> SHOW ANALYSIS /CLEAR/BRIEF file /PRINT
```

The `/CLEAR` switch clears buffer and event counters. If a filename is specified, output is written to this file. An example of output is

```
 Data Analysis [GOOTST.][GOOLIB.EXE]MGOOANL -------  8-MAR-1988 17:15:11.91
 ------- MAILBOX input --------------------
Mailbox:  , Buffer size: 8192
Status : CLOSED
```

```
 ------- DECnet input --------------------
Buffer size: 8192, Status : CLOSED
 ------- Event statistic -----------------
Events to process  :         0 , per buffer         :         0
First event to pr. :         1 , Events processed   :         0
Events per buffer  :         0 , Events skipped     :         0
Event loop execution: X$ANAL - Dyn.List -
Input event: DB:[DATA]EVENT, Output event: DB:[DATA]NEWEVENT
```

The current file headers are displayed by

```
GOOSY>SHOW ACQUISITION /INFILE/OUTFILE
```

On a separate terminal one can display a continuously updated overview by DCL command

```
 $ GSTAT env /$ANL       ! Offline
 $ GSTAT env /$TMR/$ANL   ! Online
```

Chapter 23

GOOSY Analysis

23.1 Introduction

The GOOSY Analysis Manager described in the previous sections controls the data I/O and certain analysis parameters like names of Data Elements used to store events or names of routines (un)packing events to(from) buffers. In this chapter the event loop execution is described in more detail.

23.1.1 Event Loop

The event loop performs the following steps:

1. Clear preset and execution bits for spectra and conditions.
2. Call unpack routine depending on buffer Type.
3. Call user analysis routine (optional).
4. Call Dynamic List Executor (optional).
5. Fill output event into output buffer (optional).

Figure 23.1 on page 305 shows the steps performed in the event loop. First an event is copied from the input buffer to the Data Base. It may be expanded during this step. Then a user analysis routine is called (optionally) which may access the event in the Data Base and other Data Elements. It may write new values into a Data Element for output. Then the Dynamic List Executor is called. After that (optionally) the output Data Element is copied into the output buffer which is delivered to the output channels.

23.1.2 Analysis

The user specific analysis can be done by two methods:

1. Dynamic lists (analysis tables)
2. Analysis routines

Using the first method, the analysis is determined by tables which can be modified by commands. The second method requires a user written analysis procedure. Both methods can be combined. The analysis routine is called first in this case. Then the dynamic list executor is called for execution of all dynamic lists attached. A dynamic list is required for scatter plots. The execution of the analysis routine and the dynamic list executor can be switched ON or OFF by commands for a running analysis.

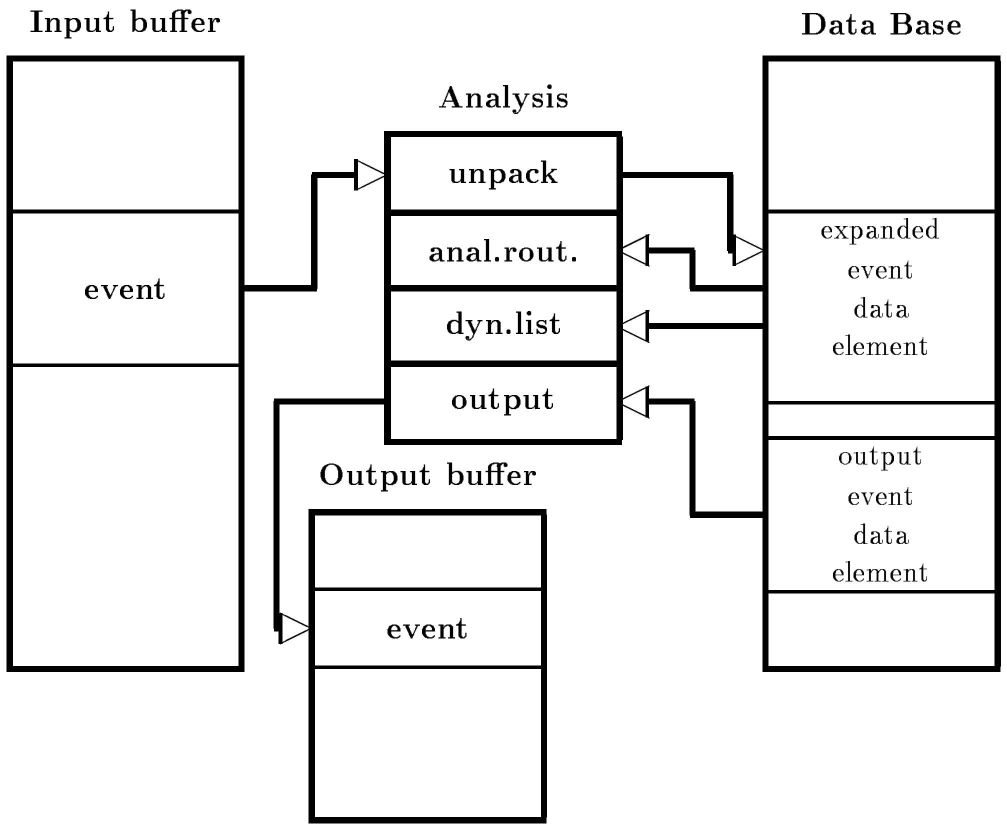


Figure 23.1: Moving events between I/O buffers and data bases in the event loop.

23.1.3 Getting the Data

As shown in figure 23.1 on page 305 the events packed in the buffer must be transferred to a Data Element in the Data Base. The analysis routines and the dynamic list executor can access that Data Element. The module transferring one event from the buffer to the Data Base is called unpack routine. Similar, the routine transferring one Data Element into an output buffer is called pack routine. Both types of routines must know about the data structures of the event, of the buffers and of the event Data Element. The analysis routine knows only the Data Element in which the event is stored by the unpack routine. All GOOSY buffers and events in these buffers are qualified by two numbers, the type and subtype. In the event loop these types are checked and the appropriate unpack routines are called. GOOSY provides several unpack and pack routines for standard buffer and event types. The user may, however, write his own unpack or pack routines, if he wants to use unsupported data formats.

23.1.4 Buffer and Event Types

GOOSY listmode data files contain buffers. The buffer and event headers are SA$bufhe and SA$evhe, respectively. The declarations can be found in the text library GOOINC. The third and fourth word in a buffer or event header specify the type and subtype of the buffer or event, respectively. Presently the following buffer types are defined. The numbers are defined in the text library GOOINC($BUFREP). Note the difference of the event structure in a buffer or in a data base.

1. **Buffer type=2, subtype=1, event type=1, subtype=1**
   Buffers formerly used by the J11 based single CAMAC crate system. The J11's write now buffers of type 4, subtype 1 (see below). There is an unpack routine A$UPJ11 for these buffers. The structure of the event data element is SA$e1_1, which is found in the text library GOOTYP. These buffers are presently 8192 bytes long.

2. **Buffer type=3, subtype=1,2, event type=3, subtype=1,2**
   Buffers from analysis output in /COMPRESS mode. On output events are compressed by A$PACMP. The subtype specifies the compression mode. On input the events are expanded by A$UPCMP. The structure of the event data element is free. The pack routine stores the full data element into the output buffer behind an event header. On input the unpack routine copies the event without header from the buffer to the event data element.

3. **Buffer type=4, subtype=1, event type=4, subtype=1,2**
   This type is used by the J11 single crate system and by analysis output in /COPY mode. The event data element structure is SA$EVENT, which is found in text library GOOTYP. The subtype 1 stands for uncompressed events, subtype 2 for zero suppressed events. Analysis output event data elements are copied by A$PAEVT into the output buffer. These events are unpacked during input by A$UPEVT. The structure of the event data element is free, but must have a standard event header. These buffers are presently 8192 bytes long.

4. **Buffer type=6, subtype=1, event type=6, subtype=1**
   This type is used by the standard MBD system. The event data element structure is SA$MBD, which is found in text library GOOTYP. These buffers are presently 8192 bytes long.

5. **Buffer type=10, subtype=1, event type=10, subtype=1,2,2..**
   This type is used by the standard VME system. The event data element structure is SA$VE10_1, which is found in text library GOOVME. The events are composed by subevents (Structure SA$VES10_1 in GOOVME). CAMAC and Fastbus structures are provided. These buffers are presently 16384 bytes long.

In the `START INPUT MAIL` command the appropriate buffer size must be specified! The default of 8192 matches the current default systems.

23.2 ONLINE Analysis Design

23.2.1 Analysis Structure

The structure of the analysis is very dependent on the experiment. The following suggestions have to be adapted to the specific situation.

1. Write a simple analysis accumulating single spectra in a dynamic list. You may use simple window conditions, too, if necessary.
2. Write a simple analysis routine which does only the things which are absolutely necessary to control the experiment.
3. Write the full analysis in a different routine.
4. A large analysis routine should be split. All routines must have the same structure as the X$ANAL routine. They must locate spectra, conditions and Data Elements they use. Their initialization entries must be called in the initialization of X$ANAL.
5. Link two analysis programs. Names others then MGOOANL can be specified in the `LANL` command and in the `CRENVIR` command. Start the simple analysis together with the data acquisition (TMR). Start the complex analysis in a different environment.
6. Use first the dynamic list only. Disable the calling of analysis routines by `SET ANAL/NOANAL`.
7. If everything looks allright, enable the calling of the analysis routine by `SET ANAL/ANAL`. Detach the dynamic list first.

23.2.2 Debugging

The analysis routines have write access to most parts of the Data Base. Therefore addressing errors may destroy valid information in the Data Base. These errors are in most cases caused by three situations:

1. The argument list of a macro is wrong. This cannot be detected in any cases by the compiler.
2. The index of a spectrum or condition array is out of bounds.
3. Some variable has no default value.

If the analysis program reports errors like 'access violation' or if the Data Base commands result in strange effects, the analysis routine should be compiled with the debug switch. The analysis process should be created with the debug switch. A break point should be set on exception. Example:

```
$ COMP X$ANAL /DEB/COM
$ LANL X$ANAL /DEB
$ GOOSY
GOOSY> DEL PROC $ANL
GOOSY> CRE PROC MGOOANL $ANL /DEB
     .....
DBG> SET BREAK /EXCEPTION   !set break point
DBG> <PF1><PF3>             !enable screen scrolling
DBG> GO
GOOSY> START ....
```

When an error occurs, the debugger stops execution and one may examine where the error occurred:

```
GOOSY>
SLEEP                       !set GOOSY prompter sleeping
DBG> <PF3>                  !refresh debug screen
DBG> SHO CALL               !shows the module and line of the error
DBG> SET MODULE modulename  !should enter the text of the module
DBG> TYPE number            !move text window to line number
```

Now you can inspect variables which may have caused the error. To return to GOOSY, one normally aborts the analysis (assume an error was found) and wakes up the sleeping GOOSY prompter:

```
DBG> EXIT
^C
GOOSY>
```

Now you may correct the analysis routine and start the whole procedure again.

23.3 User Routines

The analysis program is different from other GOOSY components in that it must call in most cases user written routines. There are in general two methods to do that.

1. The routines are called by fixed names. In this case GOOSY provides default routines, but the user may link his own analysis program with his own routines. The advantage is that no special linking is required. The disadvantage that the routine names cannot chosen freely. Presently the following routines are called:

```
X$EVENT(P_buffer)     !User unpack routine for MBD user buffers
X$ANAL()              !User analysis routine
```

By a naming convention the initialization entries have fixed names:

```
$XEVENT(P_event,L_lenght) !Initialize user unpack routine for MBD buffers
$XANAL()                  !Initialize user analysis routine
```

Note that the initialization entries may be called several times, i.e. by the commands ATTACH ANALYSIS and INITIALIZE ANALYSIS!

2. The routines are called by arbitrary names. In this case the routines must be loaded after the startup of the analysis by commands. The disadvantage is that these routines must be linked in a sharable image. This can be done by DCL command `LSHARIM` on page 296.

The user can decide which method is most suited for his purpose. In all cases the routines must follow the some rules.

23.3.1 Initialization

Each routine must have an initialization entry. This may be a PL/1 entry or a separate module which must be, however, in the same file. This entry or module is called once during the startup of the analysis or by special commands (`INITIALIZE ANALYSIS` or `ATTACH ANALYSIS`). All GOOSY Data Elements used in the routine must be located here. This is done using the $LOC macros. These macros return pointers for the specified Data Elements. For spectra and conditions these pointers are normally not used by the user, but rather by the $ACCU and $COND macros. Other initializations may be done. Note that variables set in the initialization module must be declared STATIC. If the initialization module is not an entry of the executing module, these variables must be declared outside the PROCEDURE blocks.

Argument Lists

The unpack and pack routines are initialized with the following arguments:

1. POINTER
   This pointer is NULL during startup. It is set by the `SET EVENT INPUT` or `SET EVENT OUTPUT` command and points then to the Data Element specified with these commands.

2. BIN FIXED(31)
   This longword is 0 during startup. It is set by the `SET EVENT INPUT` or `SET EVENT OUTPUT` command to the length in bytes of the Data Element specified with these commands.

By this mechanism the initialization entries may locate a default event Data Element, if the argument pointer is NULL. If it is not NULL, they can use this pointer to access the Data Element.

The other routines are called and initialized without arguments. The following section will give an overview.

23.3.2 Routine Classes and Arguments

In any case the user written routines are called by GOOSY. This means that the functions and argument lists of the routines are defined by GOOSY. Presently there are six classes of user routines. All of these must have initialization entries.

1. **Buffer unpack routines**. Argument: Pointer to input buffer.
   Initialization entry arguments: Pointer to event Data Element and length.
   These routines are called at the beginning of the event loop. They should copy one event from the buffer into an event Data Element. They must keep the event position in the buffer from one call to the other. If they return the status code XIO_NOMOREEVENT, they are called immediately with a new buffer pointer.

2. **Event unpack routines**. Argument: Pointer to buffer event.
   Initialization entry arguments: Pointer to event Data Element and length.
   These routines are called at the beginning of the event loop. They should copy one event from the buffer into an event Data Element. The difference to the buffer unpack routines is, that the pointer points to the event in the buffer rather than to the buffer itself. If they return status code XIO_SKIPEVENT, they are called immediately with a pointer to the next event in the buffer. If there is no more event, the next buffer is used.

3. **Pack routines**. Argument: Pointer to output buffer.
   Initialization entry arguments: Pointer to event Data Element and length.
   These routines are called at the end of the event loop, if output is enabled, and if no previous routine returned a status code XIO_NOOUTPUT. They copy a Data Element into the output buffer. They must keep the position of the next event in the output buffer. If there is no more space for the event, they return status code XIO_BUFFERFULL. In this case they are called immediately with a new buffer pointer. Then the pending event can be copied.

4. **Analysis routines**. No Arguments.
   Initialization entry arguments: No arguments.
   These routines are called after the unpack routines. If they return status code XIO_SKIPEVENT,

further execution of the event loop is skipped. Status code XIO_NOOUTPUT signals that no pack routine should be called for this event.

5. **Start analysis routines**. No arguments.
   Initialization entry arguments: No arguments.
   This routine is called at the beginning of the `START INPUT` commands.

6. **Stop analysis routines**. No arguments.
   Initialization entry arguments: No arguments.
   This routine is called at the end of the `STOP INPUT` commands.

23.4 Buffer Unpack Routines

The unpack routines copy one event from the GOOSY input buffer to the event Data Element in a Data Base. This Data Element can then be referenced in the analysis routine and in the dynamic list specifications. In most cases the event has to be expanded during that copy operation. The unpacking is specific for different buffer and buffer event types. Normally each buffer event type needs a specific event Data Element structure. E.g. the buffer events of type=4, subtype=1 are copied into event Data Elements of type SA$EVENT by a standard GOOSY unpack routine. The unpack routine is called by the GOOSY Analysis Manager. It gets passed the pointer to a data buffer. It controls by the return code what GOOSY will do after the call:

```
RETURN(1);                Process event.
RETURN(XIO_NOMOREEVENT);  Provide a new buffer.
RETURN(XIO_SKIPEVENT);    Skip the event.
RETURN(XIO_NOOUTPUT);     Suppress the output of this event
                          (ignored, if output is not enabled).
```

These return codes are in PL/1 defined by

```
@DCL_MSG(XIO_NOMOREEVENT);
@DCL_MSG(XIO_SKIPEVENT);
@DCL_MSG(XIO_NOOUTPUT);
```

As long as the unpack routine does not return XIO_NOMOREEVENT, it will be called with the same buffer pointer at the beginning of the next event loop. If it returns XIO_NOMOREEVENT, it will be called with the pointer to the next buffer immediately, i.e. without any call of other routines.

The unpack routine must have an initialization entry, which is called during startup of the analysis program. One pointer and one Longword are passed to it. Both are zero for normal startup. The command

```
 GOOSY> SET EVENT INPUT de-spec.
```

calls the entry and passes the pointer and length of the specified Data Element. This entry must locate all Data Elements it needs, at least the event Data Element. Use the $LOC macro for this purpose. The macros must be declared by

```
@INCLUDE $MACRO($MACRO);
```

A template is available from the file GOO$TEST:X$EVENT.PPL. For some event types standard unpack routines are provided by GOOSY. These are described in the following.

23.4.1 Standard Buffer Unpack Routines

J11 Generated Events

For buffers written by the J11 based single CAMAC crate system a standard unpack routine is provided. It copies events from buffers into a Data Element DB:[DATA]EVENT of Type SA$EVENT declared in the text library GOOTYP. If the events should be copied to another Data Element, this can be specified by the `SET EVENT INPUT` command. It must, however, have the same Data Type SA$EVENT. This structure looks like:

```
/* ========= GSI Event header ================= */
DCL P_SA$event       POINTER;
DCL 1 SA$event         BASED(P_SA$event),
    2 IA$event_dlen    BIN FIXED(15), /* data length in words */
    2 IA$event_tlen    BIN FIXED(15), /* not used */
    2 IA$event_type    BIN FIXED(15), /* event type */
    2 IA$event_subtype BIN FIXED(15), /* event subtype */
    2 IA$event(512)    BIN FIXED(15);
/*---------------------------------------------*/
```

The event Data Element must be created in the Data Base by

```
MDBM> CREATE TYPE @GOOTYP(SA$EVENT)
MDBM> CREATE ELEMENT EVENT TYPE=SA$EVENT DIR=DATA POOL=pool
```

You may use a different structure as long as the first four words are the same. The order of readout of the CAMAC modules is defined in the CAMAC description file. The value of the first module will be copied to IA$EVENT(1), the second to IA$EVENT(2) and so on. Note that the index is NOT determined by the station number! One may create a private event declaration which is compatible with the standard one by command `CREATE PROGRAM`. This command uses a CAMAC description file as input and generates the declaration.

MBD Generated Events

For buffers written by the standard MBD program an unpack routine is provided. It copies events from buffers into a Data Element DB:[DATA]EVENT of Type SA$MBD declared in the text library GOOTYP. If the events should be copied to another Data Element, this can be specified by the `SET EVENT INPUT` command. It must, however, have the same Data Type SA$MBD. This structure looks like:

```
/* Declaration of MBD event structure 6,1 */
  DCL P_SA$MBD POINTER;
  DCL 1 SA$MBD BASED(P_SA$MBD),
2 IA$MBD_dlen    BIN FIXED(15),
2 IA$MBD_tlen    BIN FIXED(15),
2 IA$MBD_type    BIN FIXED(15),
2 IA$MBD_subtype   BIN FIXED(15),

        2 SA$MBD_C1,
3 IA$MBD_C1_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C1(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C2,
3 IA$MBD_C2_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C2(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C3,
3 IA$MBD_C3_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C3(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C4,
3 IA$MBD_C4_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C4(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C5,
3 IA$MBD_C5_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C5(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C6,
3 IA$MBD_C6_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C6(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C7,
3 IA$MBD_C7_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C7(99)    BIN FIXED(15); /* data words */
    /*------------------------------------------------*/
```

The event Data Element must be created in the Data Base by

```
   MDBM> CREATE TYPE @GOOTYP(SA$MBD)
   MDBM> CREATE ELEMENT EVENT TYPE=SA$MBD DIR=DATA POOL=pool
```

One may create a private event declaration which is compatible with the standard one by command `CREATE PROGRAM`. This command uses a CAMAC description file as input and generates the declaration. Optionally it generates the J11 programs for the readout of the CAMAC modules.

Analysis Generated Compressed Events

These events are generated by an analysis if the output has been started with the `/COMPRESS` qualifier. The output event Data Element DB:[DATA]NEWEVENT is compressed and stored together with an event header in the output buffer. The Data Element may have any structure. Zero longwords are suppressed. If events should be copied from another Data Element, it can be specified by the `SET EVENT OUTPUT` command. Reading these buffers the unpack routine copies such events from the buffer into Data Element DB:[DATA]EVENT. If events should be copied to another Data Element, it can be specified by the `SET EVENT INPUT` command. The event is decompressed to its original structure and copied back from the input buffer into the event Data Element.

Analysis Generated Events

These events are generated by an analysis if the output has been started with the `/COPY` qualifier. The output event Data Element DB:[DATA]NEWEVENT must have a standard event header:

```
DCL 1 SA$event         BASED,
    2 IA$event_dlen    BIN FIXED(15), /* data length in words */
    2 IA$event_tlen    BIN FIXED(15), /* not used */
    2 IA$event_type    BIN FIXED(15), /* event type */
    2 IA$event_subtype BIN FIXED(15), /* event subtype */
    2 IA$data          any structure;
```

It is stored in the output buffer. The event type field is set to 4, the subtype field to 1. Only the specified number of words are copied to the output buffer. If events should be copied from another Data Element, it can be specified by the `SET EVENT OUTPUT` command. Reading these buffers the unpack routine copies such events from the buffer into Data Element DB:[DATA]EVENT. If events should be copied to another Data Element, it can be specified by the `SET EVENT INPUT` command. This Data Element must have an event header as shown above. The data length field of the header determines how many bytes are copied. The unpack routine copies these events from the input buffer to the event Data Element including the header.

23.4.2 User Buffer Unpack Routine

The module name must be **X$EVENT**. It is called for buffers and events of type 1, i.e user coded MBD data. Standard MBD data (type 6) are unpacked by a GOOSY routine. This routine gets passed the pointer to a data buffer. It controls by the return code what GOOSY will do after the call:

```
RETURN(1);               Process event.
RETURN(XIO_NOMOREEVENT); Provide a new buffer.
RETURN(XIO_SKIPEVENT);   Skip the event.
RETURN(XIO_NOOUTPUT);    Suppress the output of this event
                         (ignored, if output is not enabled).
```

These return codes are defined in PL/1 by

```
@DCL_MSG(XIO_NOMOREEVENT);
@DCL_MSG(XIO_SKIPEVENT);
@DCL_MSG(XIO_NOOUTPUT);
```

The unpack routine copies one event from the buffer to a GOOSY Data Element which must exist in a Data Base. It must have an initialization entry named **$XEVENT**. This entry is called during startup of the analysis program and must locate all Data Elements it needs. Use the $LOC macro for this purpose. One pointer is passed to $XEVENT and one Longword. Both are zero for normal startup. The `SET EVENT INPUT` command calls $XEVENT and passes the pointer and length of the specified Data Element. The macros must be declared by

```
   @INCLUDE $MACRO($MACRO);
```

A template is available from the file GOO$TEST:X$EVENT.PPL.

23.5 Event Unpack Routines

Instead of being called with the pointer to the data buffer as argument, the event unpack routines are called with the pointer to the event in the buffer. This calling mode is enabled by command `SET ANALYSIS/EVENT`. The return code `XIO_NOMOREEVENT` has no more meaning in these modules. GOOSY provides default unpacking routines for both modes, buffer unpack and event unpack.

23.6 Pack Routines

Sometimes it is useful to output list mode data from an analysis program. This output is started by the command:

```
GOOSY> START ANALYSIS OUTPUT
```

Then the Data Element DB:[DATA]NEWEVENT is packed into output buffers. Two packing modes are selected with the command qualifiers `/COMPRESSED` or `/COPY`. The buffers are written to DECnet (can be read by a second analysis on a different node) and optional to a file. Output is stopped by the command:

```
GOOSY> STOP ANALYSIS OUTPUT
```

If another Data Element should be used for output, it can be specified by the command:

```
GOOSY> SET EVENT OUTPUT
```

Output is done event by event. If the analysis routine returns status XIO_NOOUTPUT, the current event is not written to the output buffer.

23.6.1 Copy Output

Copy mode is selected by

```
GOOSY> START ANALYSIS OUTPUT /COPY
```

The output event Data Element must have an event header like

```
    DCL 1 SA$event          BASED,
2 IA$event_dlen    BIN FIXED(15), /* data length in words */
2 IA$event_tlen    BIN FIXED(15), /* not used */
2 IA$event_type    BIN FIXED(15), /* event type */
2 IA$event_subtype BIN FIXED(15), /* event subtype */
2 IA$data          any structure;
```

The Data Element must exist in the Data Base. It is copied to the output buffer. The first word specifies the length in words of the data field to be copied.

23.6.2 Compressed Output

Copy mode is selected by

```
GOOSY> START ANALYSIS OUTPUT /COMPRESSED
```

Then the Data Element DB:[DATA]NEWEVENT is packed into output buffers. A standard event header is added in the buffer. Event type is set to 3, subtype 1. The type (structure) of DB:[DATA]NEWEVENT can be chosen freely by the user. The Data Element must exist in the Data Base. The whole Data Element is copied.

23.7 User Analysis Routine

Besides the dynamic analysis controlled by commands the user may write an analysis routine doing some calculations, checking conditions and accumulate spectra. This routine must be named **X$ANAL**. It is called without arguments. It must provide an entry **$XANAL** doing all necessary initializations. This entry is called during the startup of the analysis program. Typically one calls the $LOC macros to locate conditions, spectra or Data Elements to be used in the routine. See the HELP for detailed description. In the X$ANAL routine one uses $ACCU and $COND macros to check conditions and accumulate spectra. The macros must be declared by

```
@INCLUDE $MACRO($MACRO);
```

A template is available from the file GOO$TEST:X$ANAL.PPL. An example is described in the GOOSY introduction manual. It is available from the file GOO$EXAMPLES:X$ANAL.PPL.

The execution of the routine can be disabled and enabled by command:

```
GOOSY> SET ANALYSIS /NOANAL   ! disable
GOOSY> SET ANALYSIS /ANAL     ! enable
```

An analysis routine may be loaded after the startup of the analysis program. In this case the name can be chosen free. The routine (and the initialization entry) must be linked in a sharable image. This can be done by command `LSHARIM` (see page 296).

23.8 Analysis Macros

The macro calls for analysis routines can be inserted by the LSEDIT editor using the $\boxed{\texttt{F\_8}}$ key.

23.8.1 $LOC

```
$LOC(type,base,dir,name,access,datatype);
$LOC1(type,base,dir,name,lowlim,uplim,access,datatype);
$LOC2(type,base,dir,name,lowlim1,uplim1,lowlim2,uplim2,access,datatype);
```

Macros to locate spectra, conditions and Data Elements. For a one dimensional name array use `$LOC1`, for a two dimensional `$LOC2` macro. You must include a check on the successful execution just behind each $LOC macro. Examples (W means write access):

```
$LOC(SPEC,base,$SPECTRUM,spectrum,W,L); /* locate spectrum in Data Base  */
$LOC(COND,base,$CONDITION,cond,W,WC);   /* locate condition in Data Base */
$LOC(DE,base,directory,name,W,type);    /* locate Data Element           */
   Declares and returns a pointer P$_base_dir_name.
   Declares and returns a length  L$_base_dir_name.
```

```
$LOC1(SPEC,base,$SPECTRUM,spectrum,1,5,W,L);/* locate spectrum array     */
$LOC1(COND,base,$CONDITION,cond,1,8,W,WC);  /* locate condition array    */
$LOC1(DE,base,directory,name,1,2,W,type);   /* locate Data Element array */

IF ^STS$success THEN @RET(STS$value);   /* Check execution success       */
```

23.8.2 $COND

```
$COND(type,base,dir,name,result,dim,x1,x2,x3,x4);
$COND1(type,base,dir,name,index,result,dim,x1,x2,x3,x4);
$COND2(type,base,dir,name,i1,i2,result,dim,x1,x2,x3,x4);
```

Macros to execute condition checks. This macro executes polygon, window, multiwindow, and pattern conditions. Pattern conditions can be of Type `ANY`, `IDENT`, `EXCL`, or `INCL`. For a one dimensional name array condition use `$COND1`, for two dimensional the `$COND2` macro. Examples:

```
$COND(WC,base,$CONDITION,cond,result,1,x1);           /* one subwindow  */
$COND(WC,base,$CONDITION,cond,result,2,x1,x2);        /* two subwindows */
$COND(ANY,base,$CONDITION,cond,result,1,x1);          /* one subpattern */
$COND(MW,base,$CONDITION,cond,result,1,x);            /* multiwindow    */
$COND(POLY,base,$CONDITION,cond,result,2,x,y,poly);   /* polygon        */
```

In the following calls `index` is the name index.

```
$COND1(WC,base,$CONDITION,cond,index,result,1,x1);    /* one subwindow  */
$COND1(WC,base,$CONDITION,cond,index,result,2,x1,x2); /* two subwindows */
$COND1(ANY,base,$CONDITION,cond,index,result,1,x1);   /* one subpattern */
```

23.8.3 $ACCU

```
$ACCU(type,base,dir,name,incr,dim,x1,x2);
$ACCU1(type,base,dir,name,index,incr,dim,x1,x2);
$ACCU2(type,base,dir,name,i1,i2,incr,dim,x1,x2);
```

Macros to accumulate one or two dimensional spectra. For a one dimensional name array spectrum use `$ACCU1`, for two dimensional the `$ACCU2` macro. Examples:

```
$ACCU(L,base,$SPECTRUM,spec,incr,1,x1);               /* one dimension  */
$ACCU(R,base,$CONDITION,spec,incr,2,x1,x2);           /* two dimensions */
```

In the following calls `index` is the name index.

```
$ACCU1(L,base,$SPECTRUM,spec,index,incr,1,x1);        /* one dimension  */
$ACCU1(R,base,$CONDITION,spec,index,incr,2,x1,x2);    /* two dimensions */
```

23.8.4 $SPEC

```
$SPEC(type,base,dir,name,value,dim,x1,x2);
$SPEC1(type,base,dir,name,index,value,dim,x1,x2);
$SPEC2(type,base,dir,name,i1,i2,value,dim,x1,x2);
```

Macros to set channles in one or two dimensional spectra. For a one dimensional name array spectrum use `$SPEC1`, for two dimensional the `$SPEC2` macro. Examples:

```
$SPEC(L,base,$SPECTRUM,spec,value,1,x1);            /* one dimension  */
$SPEC(R,base,$CONDITION,spec,value,2,x1,x2);        /* two dimensions */
```

In the following calls `index` is the name index.

```
$SPEC1(L,base,$SPECTRUM,spec,index,value,1,x1);     /* one dimension  */
$SPEC1(R,base,$CONDITION,spec,index,value,2,x1,x2); /* two dimensions */
```

23.8.5 $DE

```
$DE(base,dir,name,member)
$DE1(base,dir,name,index,member)
$DE2(base,dir,name,i1,i2,member)
```

Macros to access members of arbitrary Data Elements. For a one dimensional Data Element array `$DE1`, for two dimensional the `$DE2` macro. Examples:

```
X=$DE(base,directory,name,member);           /* scalar */
$DE(base,directory,name,member)=0;           /* scalar */
X=$DE1(base,directory,name,index,member);    /* one dimension  */
X=$DE2(base,directory,name,i1,i2,member);    /* two dimensions */
```

23.9 Dynamic Analysis

23.9.1 Activating Dynamic Lists

Dynamic Lists are Data Elements in a Data Base. Each condition check, spectrum accumulation, or scatter plot is an Entry in a Dynamic List. The creation of Dynamic Lists and Entries should be done in the DCL command procedure building the Data Base. Dynamic List Entries are executed per event and may be created and deleted dynamically (parallel to a running analysis).

Several Dynamic Lists may be executed in one analysis program. Dynamic List execution is activated by attaching to it.

```
GOOSY> ATT DYN LIST d1
```

There may be up to ten different lists attached at the same time. If it is desired to stop the execution of one list and to start the execution of another one, one would type e.g.:

```
GOOSY> DETACH DYN LIST d1
GOOSY> ATTACH DYN LIST d2
```

Already attached lists can also be stopped and started by

```
GOOSY> STOP DYN LIST d3
GOOSY> START DYN LIST d3
```

Note that the execution order is the order of attachment. This order may be changed with the `DETACH/ATTACH` commands, but not with the `STOP/START` commands. Furthermore, the `STOP/START` sequence is much faster. The following `SHOW` command gives you information about the Dynamic Lists actually executing:

```
GOOSY> SHO DYN ATT * *   !Show all Dynamic Lists of all types
```

There is a "top" command to disable and enable Dynamic List execution at all:

```
GOOSY> SET ANALYSIS /NODYN   ! disable all Dynamic Lists
GOOSY> SET ANALYSIS /DYN     ! enable all Dynamic Lists
```

23.9.2 Related Commands

All commands are executed in the Data Base Manager or in the Analysis component.

```
CREATE DYNAMIC LIST      listname
DELETE DYNAMIC LIST      listname
SHOW   DYNAMIC LIST      listname
CREATE DYNAMIC ENTRY type listname
DELETE DYNAMIC ENTRY type listname
ATTACH DYNAMIC LIST      listname (for Analysis program only)
```

```
DETACH DYNAMIC LIST      listname (for Analysis program only)
SHOW   DYNAMIC ATTACHED  listname (for Analysis program only)
STOP   DYNAMIC LIST      listname (for Analysis program only)
START  DYNAMIC LIST      listname (for Analysis program only)
```

The following switches apply for the `CREATE DYNAMIC ENTRY` commands:

`/UPDATE` The modification becomes active immediately (also for the `DELETE DYNAMIC ENTRY` command).

`/MASTER` Valid for conditions (except multiwindow) and procedures. Master Functions are executed first of all other Entries. Master conditions are executed first of all other conditions. If a master conditions result is false, the Dynamic List execution is terminated. If the same master condition is in two Dynamic Lists, both lists are skipped, if the condition was false.

In all commands explicit defaults for Data Base, node and directories can be specified. These parameters are not included in the following descriptions:

```
DYN_DIR=default directory of Dynamic List
COND_DIR=default directory of condition
SPEC_DIR=default directory of spectrum
PAR_DIR=default directory of parameters
POLY_DIR=default directory of polygon
BASE=default Data Base
NODE=default node
```

23.9.3 Execution

Note that for conditions, spectra and picture frames specific freeze bits may be set or cleared by commands. This disables/enables the execution of individual Dynamic List Entries without modifications of the Dynamic List itself.

The Dynamic List is executed in the following order (the `CREATE DYNAMIC ENTRY` subcommand keys are given in parenthesis):

1. Master procedures (PROCEDURE /MASTER)
   Call specified user written procedures (modules in sharable images).
2. Master pattern conditions (PATTERN /MASTER)
   Execute pattern condition test, return if false.
3. Master window conditions (WINDOW /MASTER)
   Execute window condition test, return if false.
4. Master function conditions (FUNCTION /MASTER)
   Call specified user function, return if false.

5. Master polygon conditions (POLYGON /MASTER)
   Check polygon, return if false.
6. Master composed conditions (COMPOSED /MASTER)
   Execute composed condition test, return if false.
7. Procedures (PROCEDURE)
   Call specified user written procedures (modules in sharable images).
8. Pattern conditions (PATTERN)
   Execute pattern condition test.
9. Multiwindow conditions (MULTI)
   Execute multiwindow condition test.
10. Window conditions (WINDOW)
    Execute window condition test.
11. Function conditions (FUNCTION)
    Call specified function (module in sharable image).
12. Polygon conditions (POLYGON)
    Check polygon.
13. Composed conditions (COMPOSED)
    Execute composed condition test.
14. Spectrum accumulation (SPECTRUM)
    Accumulates 1-2 dimensional spectra of type L,R.
15. Spectrum accumulation indexed (INDEXEDSPECTRUM)
    Accumulates 1-2 dimensional spectra of type L,R.
16. Bit spectrum accumulation (BITSPECTRUM)
    Accumulates 1 dimensional bit spectra of type L,R.
17. Scatter plots (SCATTER)
    Send buffered scatter parameter data to displays.

23.9.4 Arrays

Spectra or conditions may be arrays. In this case an index range must be specified. All additional Data Elements must be either scalar or indexed by the same range. Ranges are specified by (lower limit : upper limit).
Examples:

```
MDBM> CRE DYN ENTRY WINDOW dlist [d]e_recoil(1:5)
          PARA=[d]$event.ener
MDBM> CRE DYN ENTRY SPECTRUM dlist [d]ener1(2:4)
           PARA=[d]$event.e(2:4) CONDI=[d]de_window
MDBM> CRE DYN ENTRY SPECTRUM dlist [d]ede(1:4)
           PARA=([d]$event.e,$event.de)
MDBM> CRE DYN ENTRY INDEXED dlist [d]ede(1:7)
           PARA=([d]$event.e,$event.de)
           INDEX=[d]a.b(1)
```

[d] is the Directory specification
The difference between windows and multiwindows is that multiwindows have only one object for all subwindows, but one result bit for each, whereas windows need one object per subwindow, but have only one result bit (set, if all subwindows are true). Multiwindows may be used as filters for spectrum array accumulation. The internal dimension of the window must match the specified index range. It may also be used for 'indexed' spectrum accumulation. Then the index of the last matching subwindow is used to select the spectrum member. In the first case, the subwindows may overlap, in the second case this normally makes no sense.

```
MDBM> CRE DYN ENTRY SPECTRUM list [d]ener1(2:4)
           PARA=[d]$event.e(2:4) CONDI=[d]m_window
! three spectrum Entries are executed
MDBM> CRE DYN ENTRY INDEXEDSPECTRUM list [d]ener(2:4)
           PARA=[d]$event.e(1) INDEX=[d]m_window
! One spectrum Entry is executed
```

[d] is the Directory specification
In both cases 'm_window' must have 3 subwindows.

23.9.5 Entry Types

PROCEDURE

Command to insert an entry with a user specified procedure call:

```
CRE DYN ENTRY PROCEDURE listname MODULE=image(module)
                                 PARAMETER=(argument list)
                                 CONDITION=cond
                                 /MASTER
MODULE        module specification as 'image(module)'. Module
              must be linked in sharable image
image         logical name of shar.image
PARAMETER     arg.list of DE-members
CONDITION     name of condition (optional)
/MASTER       master Entry
```

This Entry will call a module from a sharable image. The pointers to the Data Elements specified in the argument list are passed to the procedure.
Example:

```
CRE DYN ENTRY PROCEDURE dlist
             MOD=privshar(x$loop)
             PARA=([d]$event.z4.de(5),$event.z5)
             /MASTER
```

[d] is the Directory specification
The X$LOOP declaration must be:
`ENTRY(POINTER,POINTER) RETURNS(BIN FIXED(31))`
The sharable image must be linked by the DCL command `LSHARIM`.

FUNCTION

Command to insert an entry with a user specified condition function call:

```
CRE DYN ENTRY FUNCTION listname condition MODULE=image(module)
                                 PARAMETER=(argument list)
                                 /MASTER
MODULE        module specification as 'image(module)'. Module
              must be linked in sharable image. Is used and required only
              if not specified in condition.
image         logical name of shar.image
PARAMETER     arg.list of DE-members
/MASTER       master entry
```

This entry will call a module from a sharable image. The pointers to the Data Elements specified in argument list are passed to the procedure. The first argument, a BIT(1) ALIGNED, returns the condition result.
Example:

```
CRE DYN ENTRY FUNCTION dlist [d]cond
              MOD=privshar(x$cond)
              PARA=([d]$event.z4.de(5),$event.z5)
```

[d] is the Directory specification
The X$COND declaration must be:
`ENTRY(BIT(1) ALIGNED,POINTER,POINTER) RETURNS(BIN FIXED(31))`
The sharable image must be linked by the DCL command `LSHARIM`.

PATTERN

Command to insert a pattern condition entry:

```
   CRE DYN ENTRY PATTERN listname cond PARAMETER=object
                                       /MASTER
   PARAMETER     DE-members
   /MASTER       Master entry
```

The entry will check a specified Data Element member versus a pattern. Note that four test modes can be specified with the pattern condition (IDENT, ANY, EXCL, INCL). The values of the Data Element members can be inverted bitwise. Up to 8 internal dimensions. Objects can be of type BIT(16), BIT(32), BIN FIXED(15), or BIN FIXED(31).
Example:

```
   CRE DYN ENTRY PATTERN dlist [d]main_pat
            PARA=[d]$event.pat
            /MASTER
```

[d] is the Directory specification

WINDOW

Command to insert a window condition entry:

```
   CRE DYN ENTRY WINDOW listname cond PARAMETER=object
                                      /MASTER
   PARAMETER     DE-members
   /MASTER       Master entry
```

This entry will check a specified Data Element member versus window limits. Up to 8 internal dimensions. The objects may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
   CRE DYN ENTRY WINDOW dlist [d]e_recoil
            PARA=[d]$event.ener
```

[d] is the Directory specification

MULTIWINDOW

Command to insert a multiwindow condition entry:

```
   CRE DYN ENTRY MULTIWINDOW listname cond PARAMETER=object

   PARAMETER     DE-member
```

This entry will check a specified Data Element member versus all window limits. For each check a result bit is set, which may be used to increment a spectrum array member. In addition, the number of the last matching window may be used as the index of a spectrum array member (see INDEXEDSPECTRUM). The object may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY MULTI dlist [d]e_recoil
         PARA=[d]$event.ener
```

[d] is the Directory specification

POLYGON

Command to insert a polygon condition entry:

```
CRE DYN ENTRY POLYGON listname cond PARAMETER=(x,y)
                                     POLYGON=name
                                     /MASTER
PARAMETER     DE-members for X and Y. Used and required only
              if not specified in condition.
POLYGON       Name of polygon. Used and required only
              if not specified in condition.
/MASTER       Master entry
```

It is checked whether the point X,Y is inside (true) or outside (false) the polygon. Objects may be BIN FLOAT(24), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY POLY dlist [d]poly_1
         PARA=([d]$event.de,[d]$event.ener)
         POLYG=poly
```

[d] is the Directory specification.

COMPOSED

Command to insert a composed condition entry:

```
CRE DYN ENTRY COMPOSED listname cond /MASTER

/MASTER       Master entry
```

A boolean expression of conditions is executed. The expression is specified in the corresponding condition Data Element.
Example:

```
   CRE DYN ENTRY COMPOSED dlist [d]all_ok /MASTER
```

[d] is the Directory specification

SPECTRUM

Command to insert a spectrum entry:

```
   CRE DYN ENTRY SPECTRUM listname spectrum PARAMETER=object
                                            CONDITION=cond
                                            INCREMENT=incr
   PARAMETER     DE-members for coordinates
   CONDITION     condition for spectrum (optional)
   INCREMENT     DE-member for increment (optional) (BIN FLOAT(24))
```

Supports spectra of Type BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24) with up to 2 dimensions. Coordinates can be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24).
Examples:

```
   CRE DYN EN SPECTRUM dlist [d]ener1
              PARA=[d]$event.e(1) CONDI=[d]de_window
   CRE DYN EN SPECTRUM dlist [d]ede
              PARA=([d]$event.e,$event.de)
```

[d] is the Directory specification

INDEXEDSPECTRUM

Command to insert an indexed spectrum entry:

```
   CRE DYN ENTRY INDEXEDSPECTRUM listname spectrum(l:u) PARAMETER=object
                                                        INDEX=index
                                                        INCREMENT=incr
                                                        CONDITION=cond
   PARAMETER     DE-members for coordinates
   INDEX         DE-member (BIN FIXED(31)) or multiwindow to specify
                 spectrum member
   CONDITION     condition for spectrum (optional)
   INCREMENT     DE-member for increment (optional) (BIN FLOAT(24))
```

Supports spectra of Type BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24) with up to 2 dimensions. Coordinates can be BIN FIXED(15), BIN FIXED(31) or BIN FLOAT(24). Specified spectrum must be an array. Specification of index is used to select the spectrum member to be incremented. This could be either a parameter Data Element or a multiwindow.
Examples:

```
CRE DYN EN INDEXED dlist [d]ener(1:10)
           PARA=[d]$event.e(1) INDEX=[d]m_window
CRE DYN EN INDEXED dlist [d]ede(1:5)
           PARA=([d]$event.e,$event.de) INDEX=[d]a.b
```

[d] is the Directory specification

BITSPECTRUM

Command to insert a bitspectrum entry:

```
CRE DYN ENTRY BITSPECTRUM listname spectrum PARAMETER=object
                                             CONDITION=cond
PARAMETER     DE-members for coordinates
CONDITION     condition for spectrum (optional)
```

Supports one dimensional spectra, Type BIN FIXED(31). Coordinates must be BIT(32), BIT(16), BIN FIXED(15), or BIN FIXED(31).
Example:

```
CRE DYN ENTRY BIT dlist [d]patt
          PARA=[d]$event.pat(1)
```

[d] is the Directory specification

SCATTER

This entry is inserted by the `DISPLAY PICTURE` command, if scatter frames are in the picture, or by the `DISPLAY SCATTER` command. The name of the list can be specified optionally with these commands. The default is $SCATTER. Note that this list must be attached to be active. It should be attached as the last list. Scatter Entries are deleted only by the creating display process. This may lead to 'dead' scatter Entries, i.e. if the environment name is no longer used. Attaching the list in this case a message is displayed that a link could not be opened. Then one should delete all scatter Entries of all types by the command:

```
DELETE DYNAMIC ENTRY SCATTER list * * /UPDATE
```

No scatter plot should be active during that command.

23.10 User Analysis Program

Depending on the unpack routines and the kind of analysis one can use a standard analysis program provided by GOOSY. This program can read buffers generated by the J11 single crate system, by a standard MBD system or by another analysis program. It can execute dynamic lists. By the `LOAD MODULE ANALYSIS` and `INITIALIZE ANALYSIS` commands private modules linked in a sharable image can be dynamically loaded. If one is not satisfied with this functionality, one must generate his own analysis program using private unpack and/or analysis routines. Such a specific analysis program is linked by

```
    LANL obj_list /OLB=objlib/OPT=optfile
         /CMD=cmdfile /DEB /EXE=exefile/SHARE
 e.g.:
        $ LANL X$A1,X$A2 /EXE=MYANAL1
```

Part VI

Hardware

Chapter 24

GOOSY Hardware Introduction

24.1 Front-End Equipment Supported by GOOSY

Several front-end setups (as shown in figure 24.1 on page 337) will be supported by GOOSY in future. Currently only the equipment shown inside the dashed box of figure 24.1 is available.

1. *Currently supported.*

Digitizing system	CAMAC
Readout processor	CAMAC Auxiliary crate controller type CES 2180 STARBURST, which includes the DEC J11 processor (PDP 11/73)
Event builder	CAMAC branch driver **MBD** (**M**icroprogrammed **B**ranch **D**river) by BiRa Systems Inc.
Data transmission	CAMAC branch
Software	(a) The GOOSY Transport Manager $TMR. (b) J11 stand alone program (c) MBD user code

2. *Currently supported.*

Digitizing system	CAMAC
Readout Processor	CAMAC Auxiliary crate controller type ACC 2180
Event builder	STARBURST
Data transmission	Ethernet/DECnet
Software	(a) The GOOSY Transport Manager $TMR. (b) Acquisition task and Transport Manager task on STARBURST (*user code optional*, experimental setups defined by tables).

3. *Planned.*

Digitizing system	CAMAC, FASTBUS or other.
Readout Processor	Motorola 68020 or other (?).
Event builder	VME processor DEC J11 or Motorola 68020.
Data transmission	Ethernet/DECnet and/or other(?)
Software	The GOOSY Transport Manager $TMR and software for the front-end processors (depends on processor types).

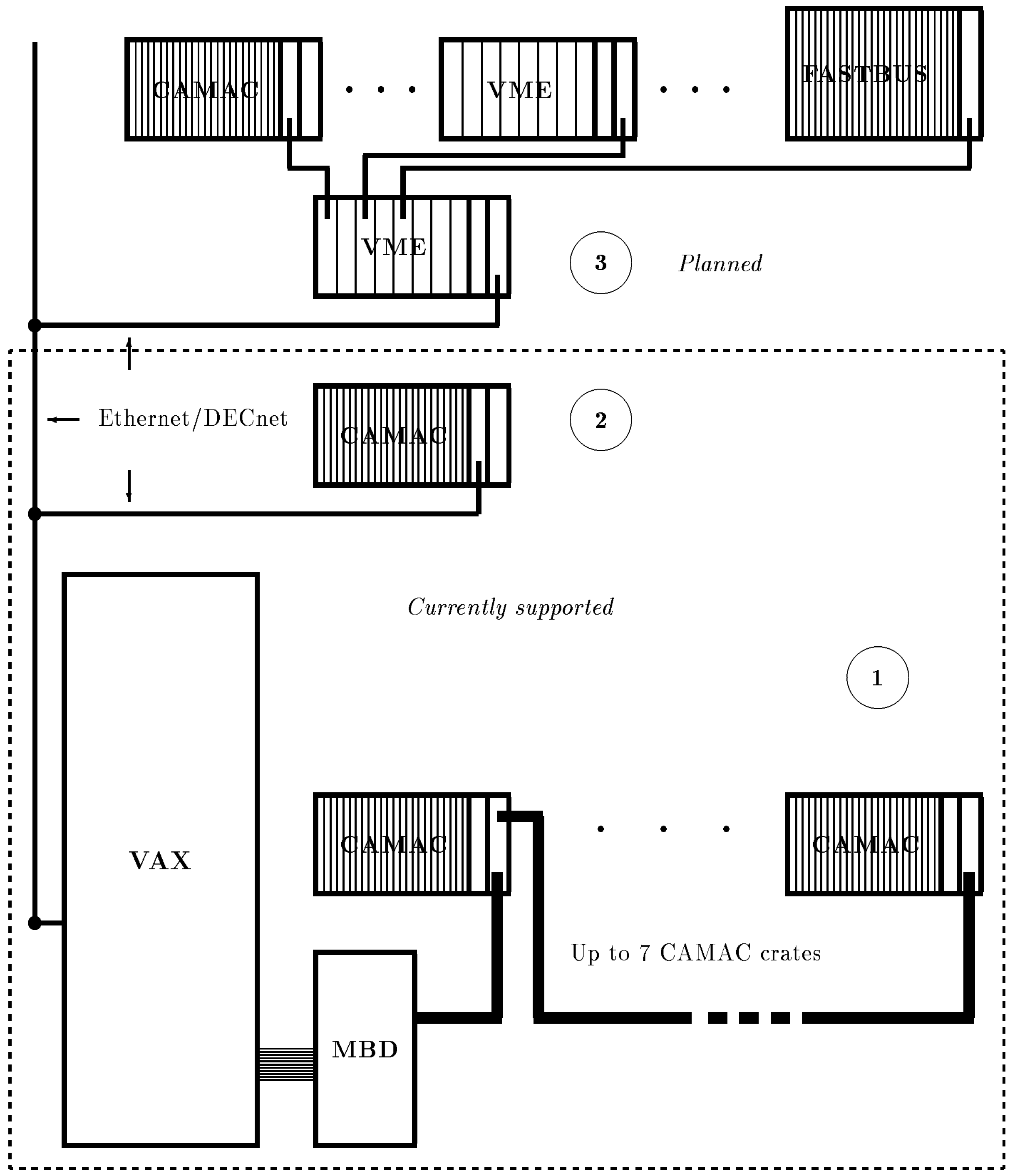


Figure 24.1: The current and the planned frontend part of the GOOSY acquisition system

Chapter 25

CAMAC Single Crate System

25.1 Introduction

For experimental set-up's which do not require more than 23 CAMAC stations and data rates up to 82 kByte/s the most comfortable solution is the J11 controlled CAMAC Single Crate System (CSCS). In this configuration the J11 microprocessor of the ACC-2180 runs under control of an operating system (RSX-11S) and acts as one node of the Ethernet/DECnet - computer network of GSI. The network facilities are used for loading the system software and the CAMAC module table as well as for the listmode data transfer to any VAX node running GOOSY. Some dedicated CAMAC modules are required:

- DM1000 Dummy crate controller located in station N=25,
- DB1000 Bootstrap module located in station N=24,
- DEQNA Q-Bus/Ethernet interface mounted inside the DB1000,
- ACC-2180 with at least 512 kByte RAM on board located in station N=23,
- GSI-IOL-box located in station N=22.

The system software supports all CAMAC modules from the GSI CAMAC pool (ADC's, TDC's, scalers...). The CAMAC readout sequence is determined by a list containing all station-subaddress combinations and the corresponding module types which belong to one listmode event. This table must be written by the user and has the form of a standard text file that can be created and modified using the VAX editor. All CAMAC modules specified will be read out in the order they occur in this table. This is also the order the parameters are stored in the corresponding event. Since the system generates a standard listmode data buffer format which cannot be modified by the user, no event unpack routine has to be provided. It is possible to enable or disable zero suppression on the fly (see: SET ACQUISITION command description). The CAMAC function F2 (read and clear) is used for readout. Since there are some ADC's which do not clear on this function (e.g. LRS 4300 ADC's), for these modules a seperate clear signal must be generated at the end of the readout period (see section 25.6.3 on page 347). For installing a new CSCS some informations have to be added to the network system data base on the VAX'es. Contact the GOOSY group for assistance (H. Sohlbach, M. Richter, tel: 494, 394).

25.2 CAMAC Module Table for the Single Crate System

In the CAMAC module table each line defines one single CAMAC module by its crate number, station number, subaddress, and type. Since presently only one crate can be addressed by the ACC 2180, the crate number is forseen only for MBD controlled systems. The module type is needed because there are some CAMAC modules which require a dedicated initialization (e.g. Silena 4418 ADC's). All modules which do not need to be initialized but must be cleared to be ready for use are summerized under the type 'STANDARD'. The following set of keywords characterizes each module:

```
CRATE=crate,                        ! crate number (presently no effect)
STATION=station,                    ! station number, range: 1 - 21
N=station,                          ! alternativ keyword for station number
SUBADDR=subaddress,                 ! subaddress, range: 0 - 15
A=subaddress,                       ! alternativ keyword for subaddress
TYP=moduletype,                     ! type of CAMAC module used (see below)
DATA=(value1,value2,value3,value4),! preset parameters for the CAMAC module
NAME=name                           ! optional free name for module
```

Comments in the table must be preceeded by an exclamation mark. The meaning of the four parameters of the 'DATA=' keyword depends upon the module type. The order of the modules in this table determines the readout scheme: the module's data are stored in the corresponding event in the same order as they are listed in the module table independent of their station numbers or subaddresses. The table can be used to create event data elements by GOOSY command `CREATE PROGRAM`. In this case the names are used as structure member names.

25.2.1 CAMAC Modules Supported by the Single Crate System

The following module types are supported:

- Type = Standard: all CAMAC modules which do not require preset values, e.g. LRS2248A, Ortec AD811, AD1000 (FIFO off), pattern units, Borer scalers ...

  ```
  Example: GOO$EXAMPLES:STANDARD.CAM
  ```

- Type = CSD24: Wenzel counter CSD24. No preset data are required, but the status register of the module is set to 512(decimal) internally whenever this type is specified.

  ```
  Example: GOO$EXAMPLES:CSD24.CAM
  ```

- Type = SI4418: Silena 4418 T/V/Q - ADC's. Presently only CAMAC readout without zero suppression is supported (status word: 2560 decimal). The J11 can be used for zero suppression. The preset values required are:

  ```
              DATA = (Offset, lower Threshold, upper Threshold, 0)
              (For subaddresses from A=0 to A=5)

              DATA = (Offset, low. Thres., upp. Thres., common Thresh.)
              (For subaddress A=6)

              DATA = (Offset, low. Thres., upp. Thres., status word)
              (For subaddress A=7, status word: 2560)

  Example: GOO$EXAMPLES:SI4418.CAM
  ```

  For more information refer to the Silena 4418 ADC manual.

- Type = LRS4300: LRS 4300 Fera 16 Channel ADC. The module table for this device must contain the pedestal values for each channel in the first data word. The second word for subaddress 0 is interpreted as status word. Presently only the status word 3072 (decimal) is supported.

  ```
              DATA = (pedestal,status word,0,0)
              (For subaddress A=0, status word: 3072)

              DATA = (pedestal,0,0,0)
              (For subaddress A=1 to A=15)
  Example: GOO$EXAMPLES:LRS4300.CAM
  ```

- Type = AD1000: AD1000 ADC adapter in connection with PU1000/PU1001 pattern unit. If you specify an AD1000 adaptor as a STANDARD module type, it will be used in the FIFO off mode. If you declare it as AD1000 explicitly, the FIFO will be enabled.

```
Caution: since the IOL-box which serves as LAM source in the CSCS does not
         contain a FIFO, in any case a PU1000/PU1001 pattern unit  m u s t
         be used to generate the LAM Input for the IOL-Box as well as the
         interrupt signal for the J11.
```

  `Example: GOO$EXAMPLES:FIFOON.CAM`

- Type = PU1000: PU1000/PU1001 pattern units with FIFO on. If the FIFO option is not needed the PU1000 can be declared as a STANDARD type unit (see above).

  `Example: GOO$EXAMPLES:FIFOON.CAM`

There is a version of the CSCS software available which supports one additional module, the LRS 2261 image chamber analyzer. If the user needs support for this module he should contact the GOOSY group for more information.

25.3 Commands to Operate the CAMAC Single Crate System in list mode

The J11 communicates with the GOOSY Transport Manager using DECnet/Ethernet. At the very beginning a connection between $TMR and the J11 must be established. This is done by the `INI ACQUISITION` command:

```
GOOSY> INI ACQUISITION NODE=node /J11
GN_J11B____$TMR: Link to J11B::CMDERR() opened
GN_J11B____$TMR: Link to J11B::TMR11S() opened
```

where 'node' means the DECnet node name of the CSCS in question. In the next step the CAMAC module table is loaded:

```
GOOSY> LOAD J11 J11B.CAM
```

The J11 now initializes all CAMAC modules and checks their X-response. When one of the modules in the table does not respond an error message will report the station number and subaddress of the module in question. After correction of this error (exchange of the CAMAC module or skipping of the module in the table) the acquisition can be started. By default the data are not compressed by the J11. If a zero suppression is desired, it can be enabled during acquisition by the command

```
GOOSY>SET ACQUISITION/COMPRESS
```

and disabled by typing

```
GOOSY>SET ACQUISITION/NOCOMPRESS
```

With zero suppression enabled two 16-bit words are transfered to the VAX for each nonzero parameter. It should be emphasized that zero suppression does only make sense if on average more than 50 percent of all parameters are true zeros. One can use the TYPE EVENT command to verify wether or not this is the case. At the very end of the session the links between $TMR and J11 must be disconnected:

```
GOOSY> STOP ACQUISITION /ABORT
GN_J11B_$TMR: Link to J11B::CMDERR terminated
GN_J11B_$TMR: Link to J11B::TMR11S terminated
```

25.4 Listmode Event and Buffer Format

The event format written by the CAMAC Single Crate System can be found in GOOTYP(SA$EVENT).

25.5 Support of CAMAC-commands and single spectra

For executing single CAMAC commands and accumulating single spectra using NIM - Adc's together with MR2000 - units is is only necessary to inform the Transport Manager and the Data Base Manager about the DECnet node name of the single crate system in question. This is done via the logical name

```
CAMAC_BRANCH_0.
```

For example, to connect the node named J11B, one has to define on DCL level

```
$ DEFINE CAMAC_BRANCH_0 "J11B::GESONE()".
```

Now all CAMAC commands or commands to operate single spectra are directed to the program GESONE running on node J11B. The name of program is unique among all single crate systems. The communication between the CSCS and the GOOSY Transprot Manager (TMR) and Data Base Manager (DBM) is done via DECnet links. Each TMR or DBM will open one DECnet link to GESONE on the CSCS. GESONE is abel to accept up to 3 DECnet links. All further link requests will be rejected. It is not necessary to stop the listmode data acquisition when executind single CAMAC - or MR2000 related commands.

25.6 Trigger Logic for the CSCS

Since only a subset of the functions needed for triggering are realized in the present trigger module, the GSI IOL-Box, the remaining functions have to be performed using NIM modules such as gate generators and coincidence units. A more complete trigger module is in preparation. The basic signals needed for triggering are:

- the gate signal for the ADC (GADC, neg. NIM)
- a signal which stays active during the ADC conversion time (TADC, neg. NIM)
- a signal that starts the ADC readout (LAM, neg. NIM and TTL inverted, width: 200 ns - 1000 ns)
- a signal which stays active as long as the J11 is busy reading the ADC's and formatting the events and buffers (BUSY, neg. NIM)
- a signal wich gets active when the system is in STOP status and not active in the RUN status (STATUS, neg. NIM).

The basic trigger logic now has to make sure that whenever GADC, TADC, BUSY, or STATUS is active no further GADC or LAM signal is passed to the ADC's, the IOL-Box or the J11. BUSY and STATUS are delivered by the J11 and appear at the corresponding front panel outputs (BF, SF) of the IOL-Box. When no RASMO unit in connection with AD1000 adaptors and PU1000 pattern units are used, the GADC, TADC, and LAM signals have to be provided by the user.

25.6.1 CSCS with RASMO Unit, FIFOs Disabled

The trigger logic has two parts, one part is the direct RASMO cabeling, the other part is the connection of RASMO and IOL-Box. In the next part the RASMO cabeling is described, those who are familiar with RASMO usage may skip this part.

- Connect the DEAD TIME output of the ADC adapter AD1000 to the corresponding DEAD TIME input on RASMO. If you want to trigger RASMO with any other signal (e.g. CAMAC TDC Start) connect it to any free DEAD TIME input on RASMO. Make sure that this trigger signal will be longer than the COINCIDENCE TIME chosen on the RASMO rotary switch and shorter than the INHIBIT output of RASMO ! (Levels: 0V = ADC free, + 3V = ADC busy = trigger).
- Connect the DATA READY output of the ADC adaptor AD1000 to the corresponding DATA READY input on RASMO. (Levels: 0V = quiescent state, +3V = ADC ready).
- Connect the RASMO DATA READY output to the corresponding input of the PATTERN UNIT PU1000/PU1001.
- Repeat this for all AD1000 adaptors.
- Switch the unused RASMO inputs to OFF.
- Connect the upper INHIBIT OUT from RASMO to the INHIBIT input of any ADC adaptor AD1000. (Level: 0V = ENABLE, + 3V = INHIBIT).
- Connect the RESET output of the last ADC adaptor AD1000 to be read out to the RESET input on RASMO. The last unit in the CAMAC module table should always be an AD1000. (Level: 0V = quiescent state, + 3V = RESET).
- Connect one LAM OUTPUT of RASMO to the NIM input of a level adaptor (e.g. LA 8000)
- Connect the TTL output of the level adaptor to the STROBE input of any ADC adaptor AD1000 (Level: 0V = quiescent state, +3V = data STROBE)
- Connect the TTL (inverted) output to the EXT INT 1 input on the front panel of the ACC-2180
- Connect the NIM output of the level adaptor to the LAM input (LI) of the IOL-box
- Connect the second LAM OUTPUT of RASMO to the CLOCK input of the pattern unit PU1000/PU1001. If you are using more pattern units you have to connect a CLOCK input to each of them. These CLOCK signals must be given before or equal to the LAM OUTPUT of RASMO (LAM comes after conversion time). (LAM output level: 0V = quiescent state, - 800mV = data STROBE, CLOCK input trigger: + 3V or/and - 800mV, i.e. fast or/and slow NIM).

- Set the rotary switch 'MAX. CONV. TIME' to the appropriate position (Rise Time Protection + ADC Conversion Time). For information on the conversion times refer to the ADC-manuals.
- Set the rotary switch 'COINCIDENCE TIME' to the desired value.
- Set the toggle switch at the upper left corner of RASMO to the appropriate position: NO CAMAC ADC's / WITH CAMAC ADC's.
- Connect the Busy output of the IOL-Box (BF) to the A input of a coincidence unit.
- Connect the Status output of the IOL-Box (SF) to the B input of a coincidence unit (e.g. CO4000). Switch the coincidence unit to work as an OR-gate 'A(inverted), B(inverted), C(inverted)', and 'OUT(inverted)'.
- Connect the OVL output of the coincidence unit to the NIM input of a NIM/TTL level adaptor.
- Connect the TTL inverted output of the level adaptor to the INHIBIT input of RASMO.

The LAM output of the IOL-Box (LF) can be used to determine the deadtime corrected event rate.

25.6.2 CSCS with RASMO Unit, FIFOs Enabled

If you want to use the FIFO buffers of the PU1000/PU1001 and AD1000 - units, RASMO, the ADC-adaptors and pattern units have to be connected as described previously. In addition, you have to:

- Connect the LAM output of the PU1000/PU1001 units to the input of a gate generator (e.g. GG 8000). The output signal of the gate generator should have a minimum delay with respect to the input and a width of 500 nanoseconds.
- Connect the output of the gate generator the the input of a NIM/ TTL - level adaptor. Connect the NIM - output of the level adaptor to the LAM input (LI) of the IOL-Box. Connect the TTL(inverted) output to the External Interrupt 1 - connector on the Front Panel of the ACC-2180.
- Connect the Status output of the IOL-Box to the NIM input of a second NIM-TTL level converter. Connect the TTL(inverted) - output to the Inhibit Input of RASMO.

25.6.3 CSCS without RASMO Unit

The only difference to the situation described above is that the GADC, TADC, and LAM must now be delivered by the user. There are several ways to do so, just one possibility should be outlined here. In the following all necessary connections are described, the cable numbers noted at the left side refer to fig. 25.2 on page 350. For the functions of the NIM and CAMAC modules used here refer to the corresponding unit's manuals.

- Cable 1: Connect the ADC gate signal (GADC) to the B input of coincidence unit 1 (e.g. CO4000). Switch the unit to 'A(inverted), B(noninverted), C(inverted), OUT'.
- Cable 2: Connect one output of coincidence unit 1 to the ADC's gate input, if the adjustable width of this output is not sufficient for the analog signal in question, you may have to use one additional gate generator.
- Cable 3: Connect the second output of coincidence unit 1 to the input of a gate generator (e.g. LRS 222). Adjust the width of the gate generator's output to the sum of the ADC gate period and the ADC's conversion time. This gives the TADC signal at the gate generator's output.
- Cable 4: Connect the noninverted output of the gate generator to the A input of coincidence unit 2.
- Cable 5: Connect the busy (BF) output of the IOL-box to the A input of coincidence unit 2.
- Cable 6: Connect the status (SF) output of the IOL-box to the B input of coincidence unit 2. Switch the unit to work as an OR gate: 'A(inverted), B(inverted), C(inverted), OUT (inverted)'.
- Cable 7: Connect the overlap output of the second coincidence unit (the logical OR signal of TADC,BUSY and STATUS) to the A input of coincidence unit 1.
- Cable 8: Connect the inverted output of the gate generator used for generating the TADC signal to the input of a further gate generator (e.g. GG8000). Adjust the output width to meet the requirements for the LAM signal (200 ns - 1000 ns).
- Cable 9: Connect the LAM output of the NIM input of a Level adaptor (e.g. LA8000).
- Cable 10: Connect the neg. NIM output of the level adaptor to the LAM input (LI) of the IOL-Box.
- Cable 11: Connect the TTL-inverted output of the level adaptor to the EXT INT 1 input of the J11 (the uppermost connector on the front panel).

Now the LAM output (LF) of the IOL-Box can be counted to determine the deadtime corrected event rate. When a TDC is used the same procedure can be done using the TDC start signal instead of the ADC gate. You will find a timing diagram for the signals and schematics using GSI pool NIM modules in fig. 25.2 on page 350 and in fig. 25.3 on page 351. As mentioned previously, there are ADC's, TDC's, and scalers which do not perform a clear when they are read out by CAMAC function F2 (e.g. LRS 4300 ADC's). In such a case a clear signal at the end of the readout period has to be generated. This can be done by connecting the overlap output of coincidence unit 2 to a Fan-in/ Fan-out unit. One output of the Fan-in/Fan-out should be wired as outlined in fig. 25.2 on page 350. A second one can be inverted and fed into a gate generator, which then will give a clear pulse at the end of the TDEAD signal. The width of this pulse should be adjusted to meet the specification of the clear input of the CAMAC module in question. Due to the internal propagation delay of the gate generator, the level adaptor and the IOL-Box it is possible that there is a spike in the total dead time signal (TDEAD) originating from the time interval between the end of the TADC signal and the busy output of the IOL-Box getting active. If such a spike occurs, the anti-coincidence condition is changed for its duration. If a gate signal is passed to the coincidence unit 2 at that time, it will possibly open the ADC gate. This can cause spurious events, but only when the average time difference between two valid gate signals is short compared to the total dead time. To avoid this problem, one can do the following (refer to the dotted lines in fig. 25.2 on page 350):

- Connect the third output of coincidence unit 1 to a gate generator. Adjust the output signal width of this gate generator to the ADC's gate period plus the ADC conversion time plus an additional period which should be app. 1 μsec or longer.
- Connect the output of this gate generator to the C input of coincidence unit 2 instead of connection 4.

The additional time will keep the anti-coincidence condition stable during the total deadtime period.

25.6.4 Jumper Settings for the ACC-2180 in the CSCS

For jumper numbers and locations on the ACC-2180 motherboard refer to the ACC-2180 manual, page 42 ff. and to fig. 25.1 on page 349):

- Jumper 5.1 All RAM
- Jumper 5.2 Z disabled
- Jumper 5.3 Start to program according power-up configuration
- Jumper 5.4 Power-up configuration: Start to boot address 1777300
- Jumper 5.5.1 Internal event 50 Hz

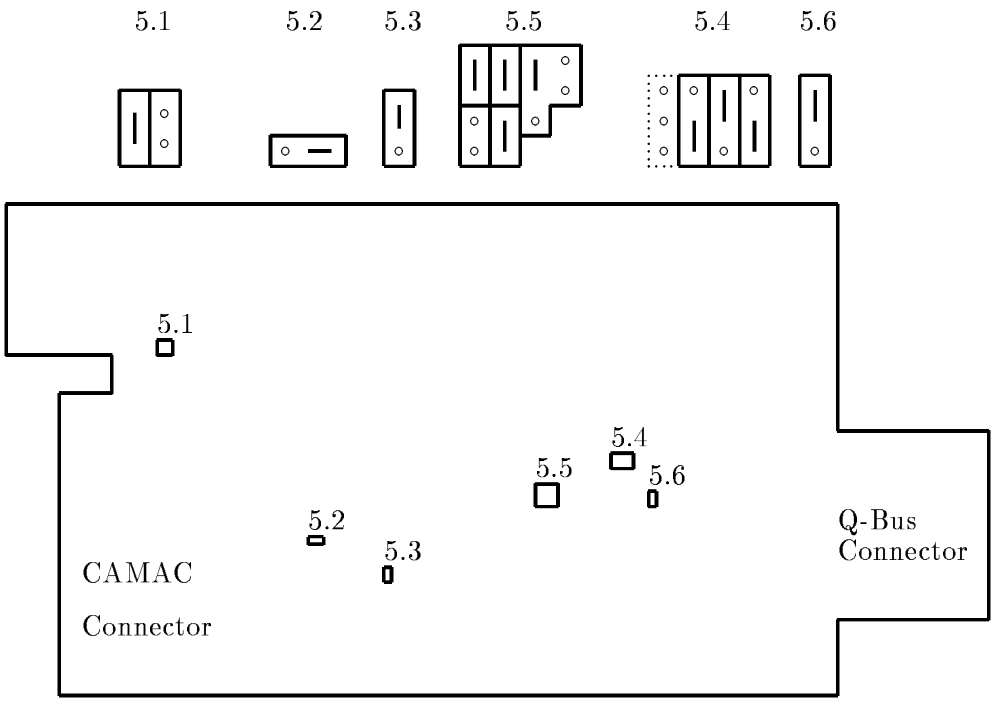


Figure 25.1: Jumpers on J11 Processor Board for **Single Crate System**

- Jumper 5.5.2 Baud rate: 9600
- Jumper 5.6 Q22 event

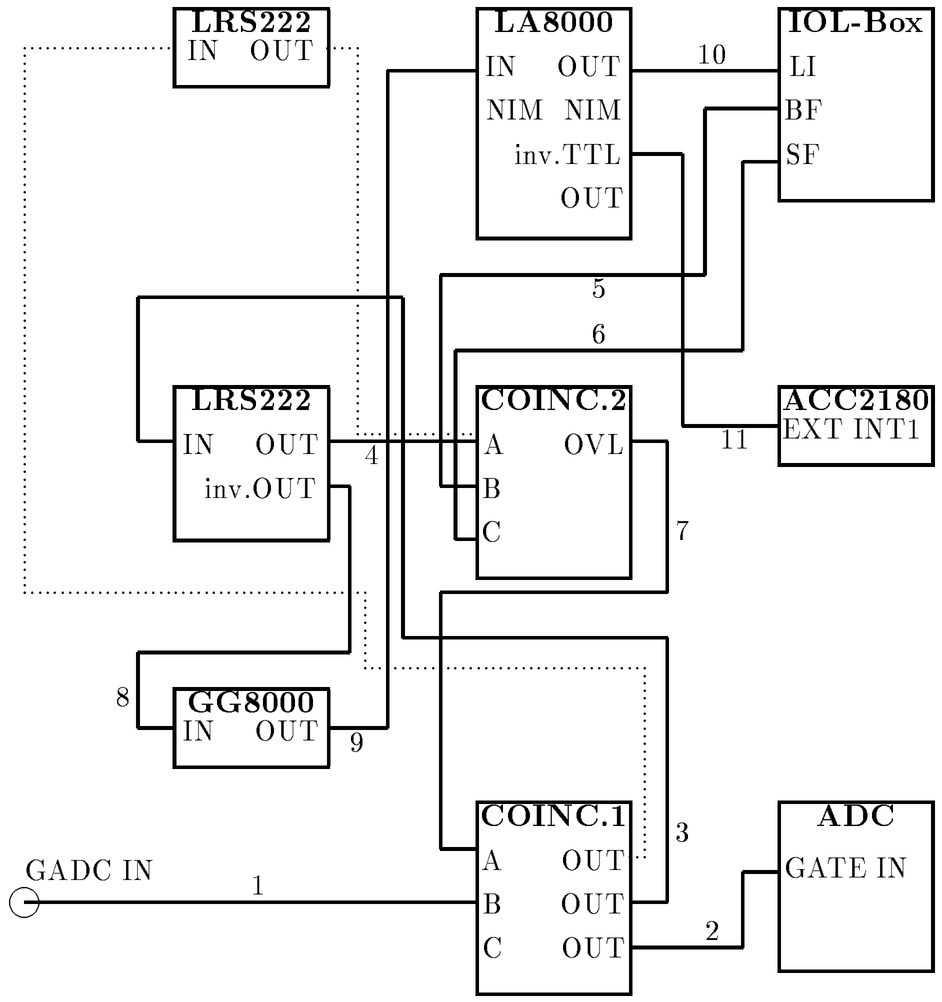


Figure 25.2: Schematics of trigger logic without RASMO

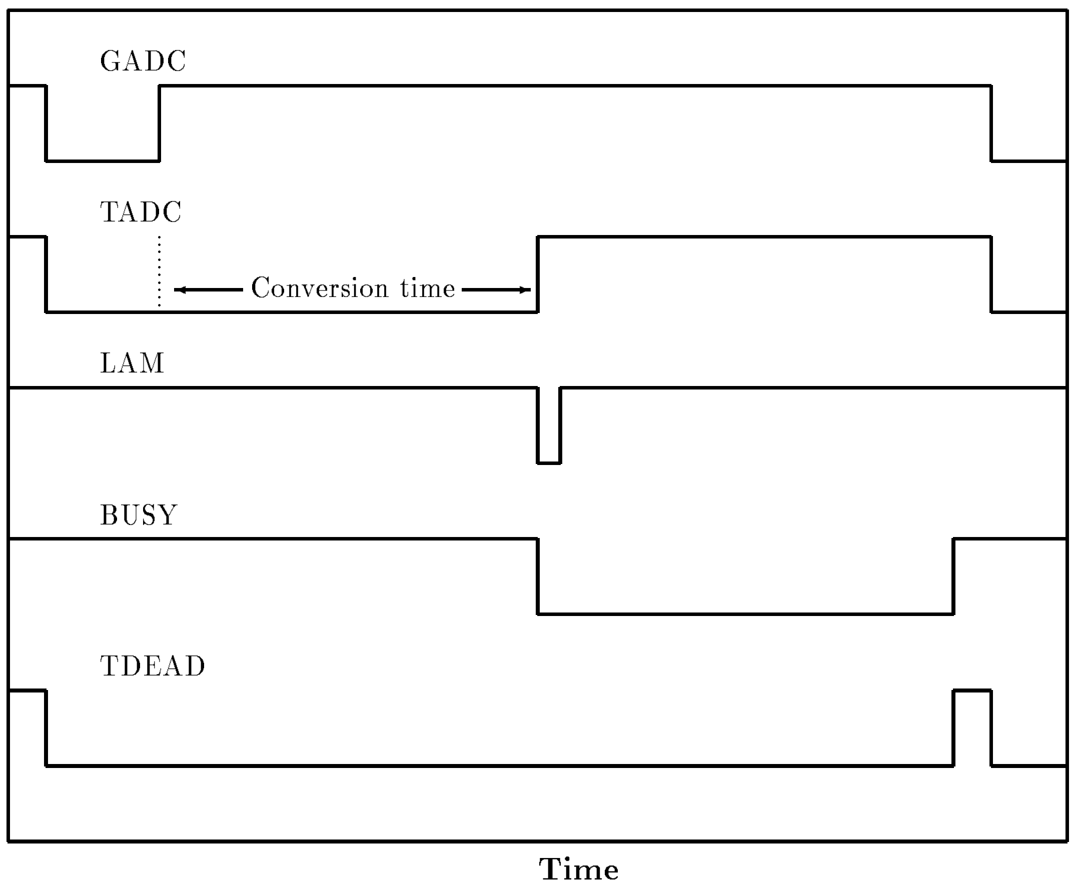


Figure 25.3: Timing diagram for trigger signals

Chapter 26

MBD Hardware

26.1 MBD Controlled Multi Crate System

The following chapters describe the more complex MBD setup which allows to use up to 7 CAMAC crates.

- Each crate is controlled by an intelligent auxiliary crate controller type CES 2180 STARBURST, which contains a "complete PDP" (but without peripheral devices): the microprocessor J11. All CAMAC addresses of the crate are mapped to the memory of the STARBURST. So the STARBURST is able to collect the data from one crate into his memory. The program in the J11 executes experiment specific modules for readout. This setup is highly flexible, because the data readout may be controlled by the data itself. Each event fills one buffer in the J11's memory.
- All crates are connected by a CAMAC parallel branch through a type A2 crate controller to the programmable branch driver MBD. The user code of the MBD reads the buffers of the J11 processors from each crate in turn for each event and builds a buffer which contains the event buffers of the single crates sequentially. If a pre-defined amount of data is collected, the whole MBD data buffer is sent to the VAX.
- The GOOSY Transport Manager $TMR receives the data buffers from the MBD, performs the list mode dump on mass storage devices (tape or disk) and enables other processes (e.g. analysis programs) on the same VAX or another VAX in the local network to pick out sample buffers from this data stream. The GOOSY Transport Manager controls all processors in the frontend equipment, performing actions like start, stop or downline loading of MBD and STARBURST. In addition, the Transport Manager may perform test procedures on single CAMAC modules or groups of CAMAC modules or single CAMAC commands using a GOOSY supplied program running on the MBD.

26.1.1 Synchronization of Processors

The synchronization of the frontend processors is performed by a CAMAC module, called the "IOL box" (**I**nput/**O**utput/**L**AM box). The following sections describe the synchronization of all processors involved in the event readout. To understand this description, refer to the functional description of the IOL box in section 26.4.1 on page 375 and figure 26.7 on page 376.

Synchronization by Hardware

The synchronization signals are illustrated in figure 26.1 on page 358).

- Each crate in our setup, which is controlled by a STARBURST, contains at station 1 (N1) an IOL box for the communication with the J11 processor (called **J11-IOL1** for crate 1, **J11-IOLn** for crate n),
- in addition, crate 1 contains at station 2 (N2) an IOL box for the communication with the MBD (called **MBD-IOL**).

MAIN TRIGGER	The **MAIN TRIGGER** is generated by the experimental electronics to start the event readout procedure. It must be inhibited during the J11 readout procedure (signaled by CINH). It will only be enabled if the START ACQUISITION command was given and the MBD has set the O2 level in its IOL box in crate 1 station 2 (enable measurement).
TRIG	**TRIG**ger. TRIG is the result of the coincidence of the $\overline{CINH}$ & START ACQUISITION & MAIN TRIGGER signals. It is delivered to the LAM input LI of each J11-IOL by a logical fanout module, e.g. a LF4000. It sets the busy flip-flop BF of the J11-IOL.
CNVWT	**C**o**n**version **W**ait. CNVWT is created from TRIG by a gate generator (e.g. LRS 222). Started by TRIG, it should have a duration of the longest conversion time in the system. It is delivered to I1 of each J11-IOL.
CINH	**C**ommon **INH**ibit. CINH is generated as the overlap signals of all busy flip-flops BF of the J11-IOL's. ORed together with the $\overline{START\ ACQUISITION}$ signal it builds the common inhibit of the system and is delivered to the experimental electronics to suppress the MAIN TRIGGER.

CRESET	**C**ommon **RESET**. CRESET is built from the trailing edge of the common inhibit CINH. It is used as common reset pulse, e.g. to clear modules, and is delivered to the LAM input LI of MBD-IOL to start the readout of all J11s by the MBD.
MBDBUF	**MBD BUF**fer. MBDBUF is created by the MBD program at O1 of MBD-IOL. It tells the J11's, which of the two internal buffers is active for the current event. It is valid only, if the MBD has evaluated the buffer address. This is guaranteed by the signal MBDWT.
MBDWT	**MBD W**ait. MBDWT is set by the LAM input at BF of MBD-IOL and removed by the MBD program. It is delivered to I2 of all J11-IOL's. It is used to serialize the access to the internal buffers of the J11 (see MBDBUF).
RESET	**RESET**. RESET is individually generated by each J11 after its readout procedure. It may be used to clear all CAMAC modules correlated with this J11.

26.1.2 Event Readout by the J11

1. The readout process is started by the trigger TRIG in each J11 at the same time. TRIG is delivered to the LAM input LI of the IOL box at station 1 (J11-IOL) in each crate.
2. If the LI was accepted, LF and BF are set and the readout process in the J11 starts.
3. The J11 immediately performs some actions to keep the interrupt logic quiet.
4. The J11 waits until the MBD has noticed, that the J11 tries to fill an event buffer, which is signalled by the termination of a NIM signal on I2 of J11-IOL (MBDWT).
5. The J11 chooses one of two internal buffers BUF1 or BUF2 for the event, depending on a signal delivered by the MBD to I4 of J11-IOL (MBDBUF).
6. The J11 writes a header to the event buffer, which contains information like crate number etc.
7. The J11 calls the user module J11EV1 which may perform the readout of modules which need no conversion time.
8. After return from J11EV1, the J11 process waits for the end of the conversion which is provided by a logical zero of the NIM signal on I1 of J11-IOL (CNVWT).
9. J11 calls user module J11EV2 which may perform readout of all CAMAC modules. Any type of selective readout is possible, but the user is responsible to write an appropriate J11 program to do such things.

10. After return from J11EV2, the J11 process checks, if the user's program modules filled some data into the event buffer. If valid data are buffered, the length of the buffer is written to the buffer header, if not, the buffer is marked to be empty.
11. The actual address and the length of the finished buffer is written to the MBD communication area in the J11's memory.
12. The busy flip-flop BF of J11-IOL is cleared and the reset pulse RESET (P2 of J11-IOL) is fired.
13. The readout procedure terminates and the J11 process waits for the next event trigger.

J11 Programs

The software in the J11 is a standard package, where 3 modules are experiment specific:

J11EVE	**J11 EV**ent **E**nable procedure. **Important note:** Determines the crate number. J11EVE is called after the start of the acquisition. May be used to initialize CAMAC modules (e.g. clear scalers etc.).
J11EVD	**J11 EV**ent **D**isable procedure. J11EVD is called after a stop of the acquisition.
J11EV1	**J11 EV**ent **1st** readout phase. J11EV1 is called before the conversion terminates (determined by signal CNVWT). May be used to read CAMAC modules, which need no conversion time (e.g. pattern units).
J11EV2	**J11 EV**ent **2nd** readout phase. J11EV2 is called after the conversion terminates (determined by signal CNVWT).

These programs can be generated by commands from a CAMAC description table. In the CAMAC module table each line defines one single CAMAC module by its crate number, station number, subaddress, and type. Since presently only one crate can be addressed by the ACC 2180, the crate number is forseen only for MBD controlled systems. The module type is needed because there are some CAMAC modules which require a dedicated initialization (e.g. Silena 4418 ADC's). All modules which do not need to be initialized but must be cleared to be ready for use are summerized under the type 'STANDARD'. The following set of keywords characterizes each module:

```
CRATE=crate,                    ! crate number (presently no effect)
STATION=station,                ! station number, range: 1 - 21
```

```
N=station,                          ! alternativ keyword for station number
SUBADDR=subaddress,                 ! subaddress, range: 0 - 15
A=subaddress,                       ! alternativ keyword for subaddress
TYP=moduletype,                     ! type of CAMAC module used (see below)
DATA=(value1,value2,value3,value4),! preset parameters for the CAMAC module
NAME=name                           ! optional free name for module
```

Comments in the table must be preceeded by an exclamation mark. The meaning of the four parameters of the 'DATA=' keyword depends upon the module type. The order of the modules in this table determines the readout scheme: the module's data are stored in the corresponding event in the same order as they are listed in the module table independent of their station numbers or subaddresses. The table can be used to create event data elements by GOOSY command `CREATE PROGRAM`. In this case the names are used as structure member names. To create the programs, use command

```
$ MDBM CREATE PROGRAM tablefile /MBD
```

For each crate specified in the file the J11 programs are generated.

26.1.3 Readout by the MBD

1. The MBD process is started by the signal CRESET.
2. The MBD inverts the MBDBUF signal on O1 which tells the J11's to choose the *other* internal buffer for the next event and prepares a new internal event buffer.
3. The MBD reads the addresses of the current event buffers in the memory of each J11 from the J11's communication areas.
4. The MBD clears the busy flip-flop BF of MBD-IOL which is used as MBD wait signal MBDWT to tell the J11's to continue 'data taking' using the *other* buffer. (That means: MBDBUF is only valid, if MBDWT is zero.)
5. The MBD copies the internal event buffers of the J11's into his internal event buffer. If a J11's buffer is marked to be empty, it is skipped.
6. The MBD checks, if its internal event buffer fits into the transfer buffer, which is delivered by the VAX. Then the internal buffer is copied to the transfer buffer or the VAX is interrupted to take the transfer buffer.
7. The MBD event building procedure terminates and MBD waits for the next event.

The software in the MBD, which performs the event readout is a standard package not changeable by the user.

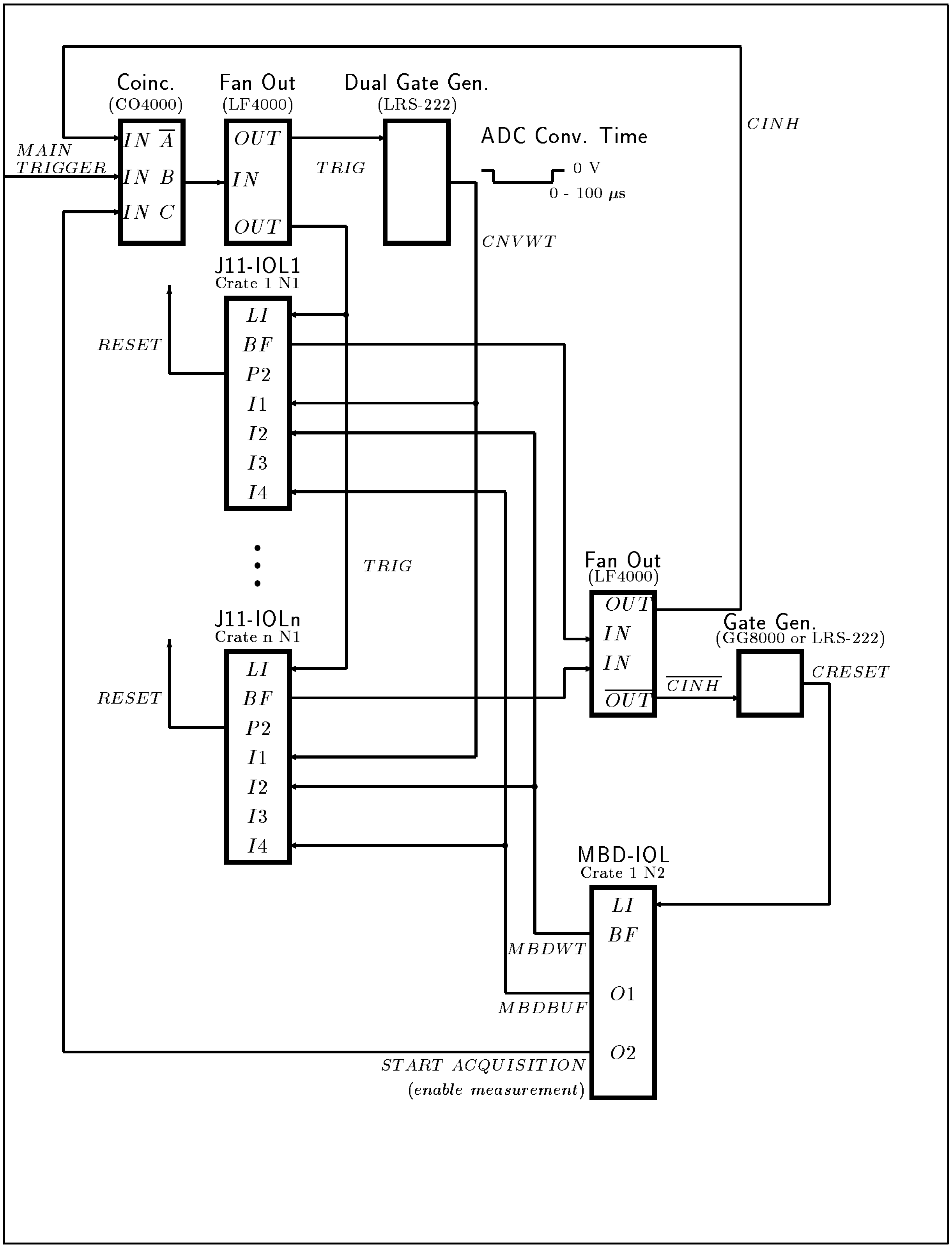


Figure 26.1: Synchronization signals

26.2 Commands for MBD and J11 Provided by $TMR

- Downline loading of programs to the frontend processors MBD and J11
- Controlling and testing CAMAC hardware
- Controlling the data acquisition

For a detailed description, refer to the GOOSY command manual and/or HELP.

26.2.1 Selecting the Frontend Processors

To have access to any MBD you first have to select a valid MBD which is not used by any other experimental group. Make sure that you are on the right VAX, mainly if you are working on DONALD or EMMA, the two cluster-VAXes in the Mess-Station. Make also sure that your CAMAC is connected to the right MBD by checking the branch cabling (e.g. DONALD MBD A = Cable DA, EMMA MBD B = Cable EB). Then enter the DCL-command:

```
$ SELECT_MBD  A              ! select the MBD A
$ SELECT_MBD  B              ! select the MBD B
```

26.2.2 Downline Loading the Frontend Processors

GOOSY provides the LOAD command for downline loading of the programs to the frontend processors.

Loading the MBD

```
LOAD MBD GOO$IO:EXEC/EXEC   ! load the MBD executive code into channel
                            ! 7 of the MBD and reset the MBD
LOAD MBD GOO$IO:ESONE       ! load the MBD with the ESONE command manager
LOAD MBD GOO$IO:Cn          ! load the MBD program 'Cn'
                            ! into the MBD. Depending on the number of crates
                            ! used load C1, C2, C3, or C4.
```

The LOAD MBD command loads the micro-code into the MBD. Currently 3 programs are running on the MBD:

1. The MBD executive code EXEC runs in channel 7 of the MBD. The executive code (EXEC) works as the partner of the VMS driver of the MBD. If any severe error occurs on the CAMAC branch the EXEC should be reloaded. If the EXEC is loaded, the MBD is reset and all other channels have to be reloaded. The EXEC code is available at GOO$IO: EXEC.BDO (object code) and GOO$IO: EXEC.MBD (source).
2. The code ESONE runs in channel 4 of the MBD. ESONE works as the partner of the Transport Managers CAMAC test features. The ESONE code is available at GOO$IO: ESONE.BDO (object ode) and GOO$IO: ESONE.MBD (source).
3. The code for the acquisition runs in channel 6 of the MBD. This code works as a partner of the Transport Managers data acquisition facility. The commands `START/STOP ACQUISITION` need the user code of the MBD. There are currently four different codes available depending on the number of CAMAC crate to be used, C1, C2, C3, and C4.

If the MBD could not be loaded because of an internal loop or any MBD hangup you may try to reset the MBD by the $TMR command

```
GOOSY> RESET MBD
```

If a new MBD program should be loaded into a previously used MBD channel the length of the new code must be shorter than or at least equal in length to the old code for this MBD channel. Otherwise you have to release the MBD channel you want to load by the $TMR command

```
GOOSY> RELEASE MBD CHANNEL chan
```

where 'chan' is the MBD channel to be released.
Be shure what you are doing when loading or resetting an MBD and check that you have selected the right MBD on the right VAX. Otherwise you may destroy running experiments.

Loading the J11/STARBURST

```
LOAD STARBURST file C=1 N=23/BOOT    ! load the task image file.TSK
                                     ! into the memory of the J11 and
                                     ! perform a boot operation
```

The `LOAD STARBURST` command loads a program, compiled with the VAX/RSX macro assembler using the macro library GOOMACJ11, and linked with the VAX/RSX task builder. This program must be written by the user. For the compilation the file must be on the current default directory. Use the DCL commands

```
$ COMPILE file.MAC                ! to compile file.MAC
$ LINKJ11 file                    ! to link the object module file.OBJ
                                  ! this produces the desired file.TSK
```

26.2.3 Controlling and Testing CAMAC Hardware

For any of the following commands the ESONE code must have been loaded into the MBD (see previous section).

CAMAC Control Commands

The command `CAMAC` is used for controlling CAMAC devices. Most of this commands affect the crate controller:

```
CAMAC DEMAND C=1 /ENABLE        ! enable demand output of crate controller
CAMAC INHIBIT C=1 /CLEAR        ! clear the Inhibit line in crate 1
CAMAC CLEAR C=1                 ! generate clear signal C in crate 1
CAMAC INITIALIZE C=1            ! generate initialize signal Z in crate 1
CAMAC SCAN C=1 N=15/STATION     ! scan all subaddresses of station 15 in
                                ! crate 1 with all function codes
CAMAC CNAF C=1 N=15 A=0 F=16 DATA=1 ! write a '1' into the register
                                ! of subaddress 0 of station 15
```

CAMAC Test Commands

A *real* problem during setup of an experiment is the test of all components of the frontend system. GOOSY provides some commands to test CAMAC modules.

```
TEST BOR_1803 C=1 N=18/FULL      ! perform a full test on a dataway display
```

Test commands are available for the following modules:

BOR_1802	BORER 1802 dataway display or compatible types (e.g. CES DSM 3320).
GSI_IOL	The GSI IOL box.
LRS_2228	LRS 2228 8 channel time to digital converter (TDC).

LRS_2249	LRS 2249 12 channel charge sensitive ADC.
LRS_2551	LRS 2551 12 channel 24-bit scaler.
LRS_4432	LRS 4432 32 channel 24-bit scaler.
LRS_4434	LRS 4434 32 channel 24-bit scaler.
REGISTER	Test any CAMAC read/write register.
SEN_2047	SEN 2047 pattern unit.

Do not stay in the $TMR menu if you use test loops!

CAMAC Access by Program for Test

It is possible to access CAMAC from a PL/I or FORTRAN program using a set of CAMAC ESONE standard procedures calls.

Be shure what you are doing! You may destroy the data of your experiment by explicit CAMAC operations.

To have access to CAMAC, you must select the MBD controller you want to use by the DCL command:

```
  $ SELECT_MBD mbd              ! mbd is A or B
```

Within your program the MBD must be assigned by the specific call

```
I$ASMBD('MBD',I_chna,I_chnb,I_chnc)
```

where I_chna, I_chnb, and I_chnc are BIN FIXED(15) variables. The CAMAC address used by the ESONE procedures is called 'external address'. This external address is built from the branch number, the crate number, the station number, and the subaddress by the procedure

```
I$CDREG(L_ext_addr,I_b,I_crate,I_n,I_a)
```

where L_ext_addr is a BIN FIXED (31) output argument which can be used in the ESONE CAMAC calls, all other input arguments are BIN FIXED(15) variables.

The following list shows the available ESONE calls. For detailed information use the Help for each module you want to use.

```
generate a dataway Clear:
 I$CCCC(L_ext_addr)

enable/disable crate demand (B_enable = '1'B to enable,
                             B_enable = '0'B to disable):
 I$CCCD(L_ext_addr,B_enable)

set/clear dataway Inhibit (B_set='1'B to set, ='0'B to clear):
 I$CCCI(L_ext_addr,B_set)

generate a dataway Initialize:
 I$CCCZ(L_ext_addr)

declare CAMAC external address:
 I$CDREG(L_ext_addr,I_b,I_c,I_n,I_a)

execute multiple CAMAC operations, L_cb is a control block with
the repeat count, the transferred count, and two unused longwords (0),
all other arguments are arrays of the repeat count length:
 I$CFGAB(I_fa,L_ext_addra,L_data_array,B_qxa,L_cb)

execute a single CAMAC operation, with function, external address,
data longword, and Q/X flags:
 I$CFSA(I_f,L_ext_addr,L_data,B_qx)

repeat the execution of a single CAMAC operation, with a function,
an external address, and a data array. The control block L_cb defines
the repetion number:
 I$CFUBC(I_f,L_ext_addr,L_data_array,L_cb)

like I$CFUBC but with an additional single CAMAC function to prepare
the repeat function, e.g. address setting:
 I$CFUBM(I_f,L_ext_addr,L_data_array,L_cb,I_fp,L_ext_addrp,L_datap,'0'B)

repeat mode block transfer. It is similar to I$CFUBC but a transfer
take place only if the CAMAC module responses with Q, otherwise the
CAMAC operation will be repeated without data transfer and without
counting down the repeat count of the control block L_cb until the Q
response is given again:
 I$CFUBR(I_f,L_ext_addr,L_data_array,L_cb)

execute a single CAMAC operation, with function, external address,
16 bit data word, and Q/X flags:
 I$CSSA(I_f,L_ext_addr,I_data,B_qx)

identical to I$CFUBC but with 16 bit data only:
 I$CSUBC(I_f,L_ext_addr,I_data_array,L_cb)

identical to I$CFUBR but with 16 bit data only:
 I$CFUBR(I_f,L_ext_addr,I_data_array,L_cb)

test crate demand enable, B_enable = '1'B if enabled:
 I$CTCD(L_ext_addr,B_enable)

test dataway Inhibit, B_inhibit = '1'B if Inhibit is set:
 I$CTCI(L_ext_addr,B_inhibit)

test crate demand (graded LAM) present, B_gl = '1'B if LAM is set:
 I$CTGL(L_ext_addr,B_gl)

test status of last CAMAC operation, L_status returns error
status codes:
 I$CTSTAT(L_status)

convert external address format to ASCII text CV_addr (CHAR(17) VAR):
 I$CVREG(L_ext_addr,CV_addr)
```

The following example gives a short impression of how to program a simple single CAMAC operation in a PL/I program:

```
%INCLUDE       I$ASMBD;        /* assign MBD            */
%INCLUDE       I$CDREG;        /* build CAMAC addresses */
%INCLUDE       I$CFSA;         /* execute single CNAF   */
DCL B_QX       BIT(2) ALIGNED; /* CAMAC Q/X return      */
DCL I_A        BIN FIXED(15);  /* CAMAC subaddress      */
DCL I_B        BIN FIXED(15);  /* CAMAC branch number   */
DCL I_CHNA     BIN FIXED(15);  /* I/O channel of MBD    */
DCL I_CHNB     BIN FIXED(15);  /* I/O channel of MBD    */
DCL I_CHNC     BIN FIXED(15);  /* I/O channel of MBD    */
DCL I_CRATE    BIN FIXED(15);  /* CAMAC crate number    */
DCL L_DATA     BIN FIXED(31);  /* CAMAC data word       */
DCL I_F        BIN FIXED(15);  /* CAMAC function        */
DCL I_N_MODULE BIN FIXED(15);  /* CAMAC module station  */
DCL L_CAMADDR  BIN FIXED(31);  /* coded CAMAC address   */
DCL 1 S_QX BASED (ADDR(B_QX)), /* structure for Q+X flags*/
```

```
      2 B_Q BIT(1),
      2 B_X BIT(1);

 /* assign MBD channel                                        */
STS$VALUE = I$ASMBD('MBD',I_CHNA,I_CHNB,I_CHNC);
IF ^STS$SUCCESS THEN DO;
  RETURN(STS$VALUE);
END;
I_B = 0;                      /* set branch number       */
I_CRATE = 4;                  /* set crate number        */
I_N_MODULE = 8;               /* set module station      */
I_A = 0;                      /* set subaddress          */
I_F = 2;                      /* set function code       */
 /* build CAMAC external address                              */
STS$VALUE = I$CDREG(L_CAMADDR,I_B,I_CRATE,I_N_MODULE,I_A);
IF ^STS$SUCCESS THEN @RET(STS$VALUE);
 /* execute CAMAC operation                                   */
STS$VALUE = I$CFSA(I_F,L_CAMADDR,L_DATA,B_QX);
 /* check on formal error, return if error occured            */
IF ^STS$SUCCESS THEN RETURN(STS$VALUE);
 /* check on CAMAC Q, return if no Q response                 */
IF ^B_Q THEN RETURN(STS$VALUE);
```

26.2.4 Controlling the Data Acquisition

During an experimental run, we need commands to start acquisition, stop acquisition, and to show what's going on. The transport manager must be initialized **once** at the beginning.

```
   INITIALIZE ACQUISITION /MBD ! initialize data acquisition
                               ! (default buffer size = 8192 bytes)
   START ACQUISITION           ! start data acquisition
   STOP ACQUISITION            ! stop data acquisition
   SHOW ACQUISITION            ! show the buffer counters
```

26.3 A Sample Experiment with MBD

To demonstrate, how to prepare an experiment, we will play with an arbitrary example.

26.3.1 Experimental Setup

Assume the following scenario:

- 5 γ detectors with anti-compton shields. We want to record the energy and the time for each event in anti-coincidence with the anti-compton shields. The name of this detector is ADAM.
- a complicated detector system with 5 segments, each segment gives a logical signal, if it was hit, and two time signals. We should record the data, if both time signals are > 0. The name of this detector is EVA.

The readout should start, if at least 1 segment of EVA was hit *and* at least 1 of ADAM's segments. Assume, that for some reasons the decision for anti-coincidence for ADAM is not possible before the trigger.

26.3.2 CAMAC Setup

CAMAC Topology

For each trigger, **both** detector systems give signals. We decide to put the digitizer for the detectors in separate crates, because our system reads the crates in parallel. This topology will speed up the readout procedure.

Zero Suppression

We decide to steer the zero suppression by logical signals. For ADAM we generate logical signals from discrimination of the analog signals for each segment, EVA delivers a signal for each segment.

Data to Collect

Now we have a summary of the signals to record:

- 5 energies of ADAM
- 5 times of ADAM
- 5 logical signals from ADAM's detector segments
- 5 logical signals from ADAM's anti-compton shields
- 5 logical signals from EVA's segments
- 10 time signals of EVA's segments (2 each segment)

26.3.3 Preparing the Hardware

We prepare two CAMAC crates:

Prepare Crate 1 for ADAM

N1 IOL box (mandatory) for synchronization of J11 in crate 1 (J11-IOL1).

N2 IOL box (mandatory) for synchronization if MBD (MBD-IOL).

N24 A2 crate controller (mandatory) must be switch to crate 1.

N23 STARBURST (mandatory, position may be changed)

N22 LAM grader (mandatory) **Important:** needs special wiring: LAM N2 to GL23!).

N3 SEN 2047 (experiment specific) pattern unit for ADAMs segment signals.

N4 SEN 2047 (experiment specific) pattern unit for ADAMs anti-compton signals.

N5 to N9 AD1000 (experiment specific) ADC adaptor for SILENA high resolution ADC's for the energy signals of ADAM

N10 LRS 2228A (experiment specific) 8 channel TDC for ADAM's time.

Prepare Crate 2 for EVA

N1 IOL box (mandatory) for synchronization of J11 in crate 1 (J11-IOL2).

N24 A2 crate controller (mandatory) must be switch to crate 2.

N23 STARBURST (mandatory, position may be changed)

N21 branch terminator (mandatory, position may be changed)

N3 SEN 2047 (experiment specific) pattern unit for EVA's segment signals.

N4 LRS 2228A (experiment specific) 8 channel TDC for EVA's first time.

N5 LRS 2228A (experiment specific) 8 channel TDC for EVA's second time.

We mount a CAMAC branch cable between the MBD and the crate controller of crate 1, and between the crate controllers of crate 1 and 2. In each crate we connect the CAMAC dataway priority signals: connect 'REQUEST' output to 'GRANT IN' of the crate controller and connect 'GRANT OUT' of the crate controller to 'GRANT IN' of the STARBURST.
On the rear of the modules the ACB flat-cable must be connected between the crate controller an the J11 and the LAM grader must be connected to the crate controller by a special black cable.

Then we do all cable connections as described in section 26.1.1 on page 354 and figure 26.1. on page 358.

26.3.4 Preparing the Software

Selecting the MBD

We make sure to be on the right VAX and we control the MBD cabling to find out which MBD we have to use. (e.g. DONALD MBD A = Cable DA, EMMA MBD B = Cable EB). Then we execute the DCL command

```
$ SELECT_MBD A              ! select MBD A
```

assuming MBD A is the right one.

MBD user code

We load the MBD executive, the ESONE program, and the file GOO$IO:C2 into the MBD, which handles 2 crates. The GOOSY commands are

```
LOAD MBD GOO$IO:EXEC/EXEC   ! load the MBD executive code
LOAD MBD GOO$IO:ESONE       ! load the MBD with the ESONE command manager
LOAD MBD GOO$IO:C2          ! load the MBD program C2
```

J11 Program

We will discuss only the program for crate 1, the reader may exercise a program for crate 2. The user modules are shown in the figures: J11EVE figure 26.2 on page 369, J11EVD figure 26.3 on page 370, J11EV1 figure 26.4 on page 370, and and J11EV2 (figure 26.5 on page 373). The declarations are shown in figure 26.6 on page 374.

The program has to be compiled by the VAX/RSX macro assembler using our general compilation DCL command COMPILE (no directory specification allowed with the program name):

```
$ COMPILE SAMPLE1.MAC  ! The program must be under the default directory
```

and linked with the VAX/RSX task builder using a special DCL command (no directory specification allowed with the program name):

```
$ LINKJ11 SAMPLE1      ! The program must be under the default directory
```

As the result the file SAMPLE1.TSK will appear on your default directory, which may be loaded to the STARBURST using the GOOSY command

```
LOAD STARBURST SAMPLE1 C=1 N=23/BOOT    ! load the task image SAMPLE1.TSK
                                        ! into the memory of the J11 and
                                        ! perform a boot operation
```

The programs J11MAIN and J11INI are available at GOO$IO and will be linked automatically. The libraries EXEMC, GOOMACPDP and GOOMACJ11 are available at GOO$LIB and will be used automatically.

```
PROC    J11EVE ; enable procedure for crate 1 (ADAM)
BEGIN
       MOV      \#1,@\#606        ; set crate number
       CAMSET   F=F2              ; set function 'read and clear'
       CAMRDS   R0,N=N3,A=A0      ; clear pattern unit
       CAMRDS   R0,N=N4,A=A0      ; clear pattern unit
       CAMSET   F=F24             ; set CAMAC function 24 to start AD1000
       CAMFNC   N=N5,A=A0         ; start ADC adaptor AD1000
       CAMFNC   N=N6,A=A0         ; start ADC adaptor AD1000
       CAMFNC   N=N7,A=A0         ; start ADC adaptor AD1000
       CAMFNC   N=N8,A=A0         ; start ADC adaptor AD1000
       CAMFNC   N=N9,A=A0         ; start ADC adaptor AD1000
       RET
ENDPRO
```

Figure 26.2: The J11 user module J11EVE

26.3.5 Testing the CAMAC Hardware

The following is a listing of a dialogue. We start a GOOSY session:

```
B:Z4JS$ CRENVIR z4/$tmr                 ! start GOOSY environment
GOOSY environment Z4 created
HVR connected from V750A::Z4______$TMR(GN_XX_PRCTRL)
---------------------------------------------------
GOOSY environment process GN_Z4 (*)
GOOSY component process   GN_Z4______$SVR
GOOSY component process   GN_Z4______$TMR
B:Z4JS$ SELECT_MBD A                    ! select the MBD (A)
B:Z4JS$ GOOSY                           ! enter GOOSY prompter
SUC: GOOSY> LOAD MBD goo$io:exec/EXEC    ! load the MBD EXEC code
   loading MBD executive, size %00166
SUC: GOOSY> LOAD MBD goo$io:esone        ! load the MBD ESONE code
   loading MBD channel 4 with base %01000 and %00325
   this program will execute for channel B
SUC: GOOSY> LOAD MBD C2                  ! load MBD data acquisition code
   loading MBD channel 6 with base %02000 and %01174
   this program will execute for channel A
SUC: GOOSY> CAMAC INITIALIZE c=1         ! initialize the crate
SUC: GOOSY> CAMAC INHIBIT c=1/CLEAR      ! clear the inhibit
Inhibit is clear in Crate        1
```

```
PROC    J11EVD    ; disable procedure for crate 2
BEGIN
        CAMSET  F=F25                ; set function code 25
        CAMFNC  N=N5,A=A0            ; stop ADC adaptor AD1000
        CAMFNC  N=N6,A=A0            ; stop ADC adaptor AD1000
        CAMFNC  N=N7,A=A0            ; stop ADC adaptor AD1000
        CAMFNC  N=N8,A=A0            ; stop ADC adaptor AD1000
        CAMFNC  N=N9,A=A0            ; stop ADC adaptor AD1000
        RET
ENDPRO
```

Figure 26.3: The J11 user module J11EVD

Note: If J11INI calls J11EV1, R4 points to the current internal buffer position

```
PROC    J11EV1       ; first readout procedure crate 1 (ADAM)
BEGIN
        CAMSET  F=F2                  ; set function code 2 'read and clear
        CAMRDS  ADADP,N=N3,A=A0       ; read detector pattern for ADAM
        CAMRDS  ADAAP,N=N4,A=A0       ; read anti-compton pattern for ADAM
        RET
ENDPRO
```

Figure 26.4: The J11 user module J11EV1

```
SUC: GOOSY> CAMAC CLEAR c=1              ! generate CAMAC clear signal
```

First we check all cables and connectors of the branch. For this reason, we place a dataway display (BORER 1802 or CES DSM 3320) in a free slot of the crate, e.g. N18:

```
SUC: GOOSY> CAMAC CNAF C=1 N=18 A=0 F=16 D=%O77777777  ! write 'ones'
 CNAF done C=1 N=18 A= 0 F=16 X:1 Q:1 data: 16777215  %XFFFFFF  %O77777777
SUC: GOOSY> CAMAC CNAF C=1 N=18 A=0 F=0                ! read back
 CNAF done C=1 N=18 A= 0 F= 0 X:1 Q:1 data: 16777215  %XFFFFFF  %O77777777
SUC: GOOSY> CAMAC CNAF C=1 N=18 A=0 F=2                ! read and clear
 CNAF done C=1 N=18 A= 0 F= 2 X:1 Q:1 data: 16777215  %XFFFFFF  %O77777777
SUC: GOOSY> CAMAC CNAF C=1 N=18 A=0 F=0                ! read again
 CNAF done C=1 N=18 A= 0 F= 0 X:1 Q:1 data:        0  %X000000  %O00000000
SUC: GOOSY> TEST BOR_1802 C=1 N=18/LOOP                ! start test
SUC: GOOSY> TEST LRS_2228 C=1 N=3/START                ! start test
SUC: GOOSY> TEST CAMAC/LIST
```

```
   Test-Name          B  C  N  state   delta  rep.count  err.count  err.limit
   BOR_1802           0  1 18   LOOP     1.0       1395          0        100
   LRS_2228           0  1  3   STRT     1.0          8          0        100
---I$CTSEM:GN_Z4______$TMR(00000090):BALDUIN(CLUSTER_B):-22-JAN-87 16:41:04.83
%GOOIO-E-CTSERR, Error in CAMAC test of LRS_2228 at C=1,N=3,A=5
                 Syndrome:Failed to convert with F(25), value:     0
---I$CTSEM:GN_Z4______$TMR(00000090):BALDUIN(CLUSTER_B):-22-JAN-87 16:41:06.71
%GOOIO-E-CTSERR, Error in CAMAC test of LRS_2228 at C=1,N=3,A=5
                 Syndrom:Failed to convert with F(25), value:     0
---I$CTSEM:GN_Z4______$TMR(00000090):BALDUIN(CLUSTER_B):-22-JAN-87 16:41:07.87
ERR: GOOSY> TEST CAMAC/LIST
   Test-Name          B  C  N  state   delta  rep.count  err.count  err.limit
   BOR_1802           0  1 18   LOOP     1.0       4380          0        100
   LRS_2228           0  1  3   STRT     1.0        121          3        100
```

We found a bad module: TDC LRS2228A in slot N3! After changing the module, we go on:

```
SUC: GOOSY> LOAD STARBURST SAMPLE1 C=1 N=23/BOOT
   Base %000000000   Size %000022100  Start address %0003300
SUC: GOOSY> START ACQUISITION
---I$ACQCM_STA_:GN_Z4______$TMR(00000090):BALDUIN(CLUSTER_B):-22-JAN-87 16:45:32.08
%GOOIO-E-AQUNOINIT, Acquisition not yet initialized
%GOOIO-E-AQUNOINIT, Acquisition not yet initialized
ERR: GOOSY> INIT ACQUISITION
SUC: GOOSY> START ACQUISITION
```

26.3.6 Test and Debugging of Software

The main debugging tool for the J11 programs is the type command. One can type the buffer as it appears for the $TMR.

```
SUC: GOOSY> TYPE EVENT 2

***************** Buffer ******************************************
Buffer type =    6 , Buffer number =       3 , Total length =    0
Subtype =        1 , Events  =           203 , Data  length = 4072
No spanning events                           , Used  length = 4072

================= Event 1    =================================
Event type   =   6 , Subtype  = 1 , Data length =         16
================= Subevent 1    ==============================
Subevent data length =      7 , CAMAC crate = 1 , Counter (8 bits) =  151
1        :7       1       0       2914    0       0       0
```

```
================= Subevent 2    ==============================
Subevent data length =      5 , CAMAC crate = 2 , Counter (8 bits) =  151
1         :5       2       0       0       0

================= Event 2    =================================
Event type   =   6 , Subtype  = 1 , Data length =          16
================= Subevent 1    ==============================
Subevent data length =      7 , CAMAC crate = 1 , Counter (8 bits) =  152
1         :7       1       0       2914    0       0       0
================= Subevent 2    ==============================
Subevent data length =      5 , CAMAC crate = 2 , Counter (8 bits) =  152
1         :5       2       0       0       0
```

The example shows two events, which are produced by our J11 program.

```
SUC: GOOSY> SHOW ACQUISITION
 ------  Status of Data Acquisition: ----------- 24-JUN-1988 17:21:46.07
Buffer size   :  8192    Count: 12
Queues: MBD:     4  LMD:  0  Current: FREE:    7  MBD:   4  LMD:   0
Acquisition : STARTED,  List mode dump: STOPPED,   File: CLOSED
Buffer- and Event-Statistic:
    MBD reads:          105 buffers      20909 events since clear
                        105 buffers      20909 events since start
    LMD write:            0 buffers since clear
                          0 buffers since start
                          0 buffers since open
----------------------------------------------------------------------
  Online Analysis statistic:
  GOOSY_GOOR_1 buffers:          1   0%,           1   0%  since clear
  GOOSY_GOOR_2 buffers:          1   0%,           1   0%  since clear
  GOOSY_GOOR_3 buffers:          1   0%,           1   0%  since clear
Enabled:  buffer check -
Disabled: MBX synchronization - NET synchronization - exclusive -
SUC: GOOSY>
```

Note: If J11INI calls J11EV2, R4 points to the current internal buffer position.

```
PROC    J11EV2    ; readout procedure for crate 1 (ADAM)
IDATA             ; following a data section
ADAM1: .WORD   1.                ; this is the first segment of ADAM
        CAMCNA  N5,A0             ; energy of 1. segment of ADAM
        CAMCNA  N10,A0            ; time of 1. segment of ADAM
           .
           .
           .
ADAM5: .WORD   5.                ; this is the 5. segment of ADAM
        CAMCNA  N9,A0             ; energy of 5. segment of ADAM
        CAMCNA  N10,A4            ; time of 5. segment of ADAM
BEGIN             ; begin readout
        CAMSET  F=F0              ; set function code 0 'read'
        MOV     ADADP,R0          ; ADAM detector pattern was read in J11EV1
        MOV     ADAAP,R1          ; ADAM anti-compton pattern was read in J11EV1
IF      NE      R0                ; if no segment fired, return
THEN
        CMP     R0,R1             ; compare patterns
BEQ     HOME             ; if all compton, return
        BEGDET
        MOV     \#1.,(R4)+         ; write detector id to buffer
        MOV     R4,R5             ; save address of multiplicity
        CLR     (R4)+             ; clear multiplicity
        IFL     NE  \#B0,ADAAP     ; check anti-compton 1. segment
        THEN
           CAMELE  R0,\#B0,\#ADAM1,2. ; move two parameters from CAMAC to buffer
        ENDIF
                .
                .
                .
        IFL     NE  \#B4,ADAAP     ; check anti-compton 5. segment
        THEN
           CAMELE  R0,\#B4,\#ADAM5,2. ; move two parameters from CAMAC to buffer
        ENDIF
        ENDDET
ENDIF
HOME:
        RET
ENDPRO
```

Figure 26.5: The J11 user module J11EV2

```
        .TITLE  SAMPLE1: read out crate 1
.MCALL  SPM
        SPM     TEST,X70886,MACLIB=<CAM>
        .MCALL  CAMELE
        .MCALL  BEGDET
        .MCALL  ENDDE
;  here place the code for J11EVE, J11EVD, J11EV1, J11EV2
; ===>
                   .
                   .
                   .
IDATA                           ; impure data PSECTion
ADADP:  .WORD   0                       ; memory location for detector pattern
ADAAP:  .WORD   0                       ; memory loaction for anti-compton
.END
```

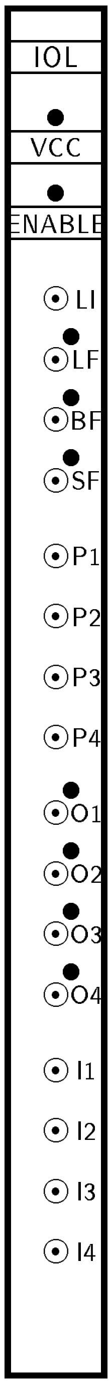
Figure 26.6: Declaration for the user modules

26.4 Hardware for MBD System

26.4.1 Functions of the IOL box

The module has 16 LEMO plugs (see figure 26.7 on page 376):

LI	*Input.* LAM input. A NIM signal put into LI, if the LAM of the module is enabled sets the internal LAM flip-flop, the output signals LF (→) and BF (→) are set.
LF	*Output.* LAM flip-flop. The signal is set by the LAM input signal LI, cleared and read by CAMAC functions (see figure 26.7 on page 376).
BF	*Output.* Busy flip-flop. The signal is set by the LAM input signal LI, cleared and read by CAMAC functions (see figure 26.7).
SF	*Output.* Status flip-flop. The signal is set, read and cleared by CAMAC functions (see figure 26.7).
O1..O4	*Output.* Output 1..4. The signal is set, cleared and read by functions (see figure 26.7).
P1..P4	*Output.* Pulse 1..4. A CAMAC function (see figure 26.7) generates a pulse of 10ns duration.
I1..I4	*Input.* Input 1..4. The state of the input signal may be read by CAMAC function (see figure 26.7).



Input/Output Signals

Signal	in/out	set	clear	read	Remarks
LI	in	-	-	-	sets LF, BF and creates LAM, if LAM enabled
LF	out	LI, F25·A0	F16·A1·W5, F10·A0	F0·A0·R5	-
BF	out	LI, F25·A0	F16·A1·W6	F0·A0·R6	-
SF	out	F16·A1·W8	F16·A1·W7	F0·A0·R7	-
O1	out	F16·A0·W1	-	F0·A0·R1	-
O2	out	F16·A0·W2	-	F0·A0·R2	-
O3	out	F16·A0·W3	-	F0·A0·R3	-
O4	out	F16·A0·W4	-	F0·A0·R4	-
P1	out	F16·A1·W1	-	-	100ns pulse output
P2	out	F16·A1·W2	-	-	100ns pulse output
P3	out	F16·A1·W3	-	-	100ns pulse output
P4	out	F16·A1·W4	-	-	100ns pulse output
I1	in	-	-	F0·A1·R1	-
I2	in	-	-	F0·A1·R2	-
I3	in	-	-	F0·A1·R3	-
I4	in	-	-	F0·A1·R4	-

CAMAC Functions

CAMAC Function	Purpose
F0·A0·R1	read O1
F0·A0·R2	read O2
F0·A0·R3	read O3
F0·A0·R4	read O4
F0·A0·R5	read LF
F0·A0·R6	read BF
F0·A0·R7	read SF
F8·A0	Test LF Q=1, if LF set
F10·A0	clear LF
F24·A0	disable LAM
F25·A0	set LF, BF, SF
F26·A0	enable LAM
F16·A0·W1	set O1
F16·A0·W2	set O2
F16·A0·W3	set O3
F16·A0·W4	set O4

CAMAC Function	Purpose
F0·A1·R1	read I1
F0·A1·R2	read I2
F0·A1·R3	read I3
F0·A1·R4	read I4
F16·A1·W1	set P1
F16·A1·W2	set P2
F16·A1·W3	set P3
F16·A1·W4	set P4
F16·A1·W5	set LF
F16·A1·W6	set BF
F16·A1·W7	set SF
F16·A1·W8	clear SF
Z·S2	clear LF, BF, SF and O1..O4

Figure 26.7: Functions of the IOL box

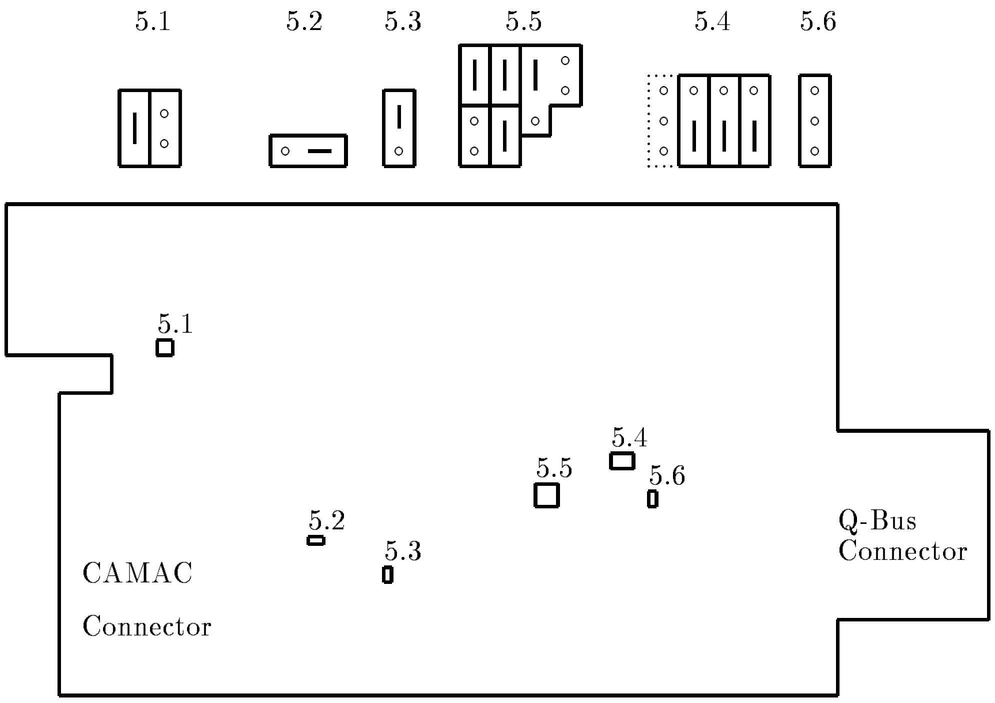


Figure 26.8: Jumpers on J11 Processor Board for **MBD System**

26.4.2 Jumper Setting for the ACC-2180 with MBD

For jumper numbers and locations on the ACC-2180 motherboard refer to the ACC-2180 manual, page 42 ff. and to fig. 26.8 on page 377):

- Jumper 5.1 All Ram
- Jumper 5.2 Z disabled
- Jumper 5.3 Start to program according power-up configuration
- Jumper 5.4 Power-up configuration: Start to Trap 24
- Jumper 5.5.1 Internal event 50 Hz
- Jumper 5.5.2 Baud rate: 9600
- Jumper 5.6 remove jumper

Part VII

Appendices

Appendix A

Using a Terminal and Editing Command Lines

On the Alpha AXP or VAX terminal all input can be typed ahead, even if the CPU does not echo. The type ahead buffer allows 80 characters to be typed ahead.

The following keys allow you to edit the current DCL command line (and the command lines of most utilities). For some keys to work, the `SET TERMINAL/LINE_EDITING` command must be in effect, which is the start default for all terminals at GSI. (Enter the `SHOW TERMINAL` command to display your terminal's attributes.)

`F12` **or** `Ctrl` H **or** `BACKSPACE` **or** `BS` — Moves the cursor to the beginning of the line.

`F14` **or** `Ctrl` A — Changes between `SET TERMINAL/OVERSTRIKE` and `SET TERMINAL/INSERT`. The start default for all terminals at GSI is the insert mode.

`Ctrl` E — Moves the cursor to the end of the line.

`Ctrl` R — Reshows the current command line.

`Ctrl` X — Clears the typeahead buffer.

`Ctrl` U — Deletes all characters to the left of the cursor.

`<X` — Deletes one character to the left of the cursor, moving the cursor one space to the left.

`F13` **or** `Line Feed` **or** `LF` **or** `Ctrl` J — Deletes the word to the left of the cursor.

$\boxed{\Rightarrow}$ **or** `Ctrl` F — Moves the cursor one character right.

$\boxed{\Leftarrow}$ **or** `Ctrl` D — Moves the cursor one character left.

$\boxed{\Uparrow}$ **or** `Ctrl` B — Recalls the previously given DCL command (the last 20 commands can be recalled in this way).

`RECALL/ALL` — Show the last 20 commands.

`RECALL n` — Gets the n^{th} command.

`RECALL xyz` — Gets the last command starting with the characters `xyz`.

$\boxed{\Downarrow}$ — Recalls the DCL command entered after the current command.

$\boxed{\texttt{Ctrl}}$ `O` — Stops and starts the screen output. The execution of a program or command procedure will *not* be stop. Any error output or the DCL prompt will automatically switch on the output again.

$\boxed{\texttt{F1}}$ **or** $\boxed{\texttt{HOLD}}$ — Stops and starts the screen output. The execution of a program or command procedure will be stopped waiting to continue with the screen output. If your terminal will not respond of any key stroke, try the $\boxed{\texttt{F1}}$ key on your terminal. This key is active for each window under DECwindows/Motif individually.

$\boxed{\texttt{Ctrl}}$ `S` — Stops the screen output. The execution of a program or command procedure will be stop waiting to continue with the screen output. Restart output by typing $\boxed{\texttt{Ctrl}}$ `Q`. This key is active for each window under DECwindows/Motif individually.

$\boxed{\texttt{Ctrl}}$ `Q` — Starts the screen output previously stopped by $\boxed{\texttt{Ctrl}}$ `S`. The execution of a program or command procedure will be stop waiting to continue with the screen output. This key is active for each window under DECwindows/Motif individually.

$\boxed{\texttt{Ctrl}}$ `W` — Rewrites (refreshes) the whole screen within several utilities using screen mode, like `LSE` or `DEBUG`.

To write special characters like an umlaut you must compose a sequence. Depending on the keyboard and the terminal press and the release the Compose key only or simultaneously press and then release the Compose key and the space bar, and then type the two characters separately which are needed for the special character. In the following is a brief list of composed characters:

ä = ”a, ö = ”o, ü = ”u, Ä = ”A, ß = ss, ÿ = ”y, á = a’, à = a‘, â = a^ , ç = c, å = a*, é = e’, è = e‘, ï = i”, ø= o/, Ø= O/, æ= a, Æ= AE, ñ = n , ©= co, ª = a_, £= L-, §= s!, ± = +-, ≪ = <<, ≫ = >>, ½ = 12, ¼ = 14, µ = /u, ^{0} = 0^ , ^{1} = 1^ , ^{2} = 2^ , ^{3} = 3^ .

There are complete tables of available characters. In DECwrite you have much more special characters including Greek in the ’Text → Special’ menu.

While using the terminal, you can have different sessions of the Terminal Server. Once you have started the first session (by logging in), you can ”break” out of your session by pressing $\boxed{\texttt{F5}}$ (or $\boxed{\texttt{Ctrl}}$ $\boxed{\texttt{F5}}$ and $\boxed{\texttt{Return}}$ on a PECAD). After the `S200xx Local>` prompt connect to any Alpha AXP or VAX or to the IBM by typing `CONNECT service` and then simply log in to the chosen service. The number of simultaneous sessions is limited by default to 4. You can have only two sessions if you use the multisession option together with a VT330, VT340, or VT420 terminal.

You can move through your established sessions by pressing `Ctrl` \, or by breaking out to the Terminal Server local mode with key `F5` (or `Ctrl` `F5` and `Return` on a PECAD) and then using the following `Local>` commands: `FORWARD` or `BACKWARD`.

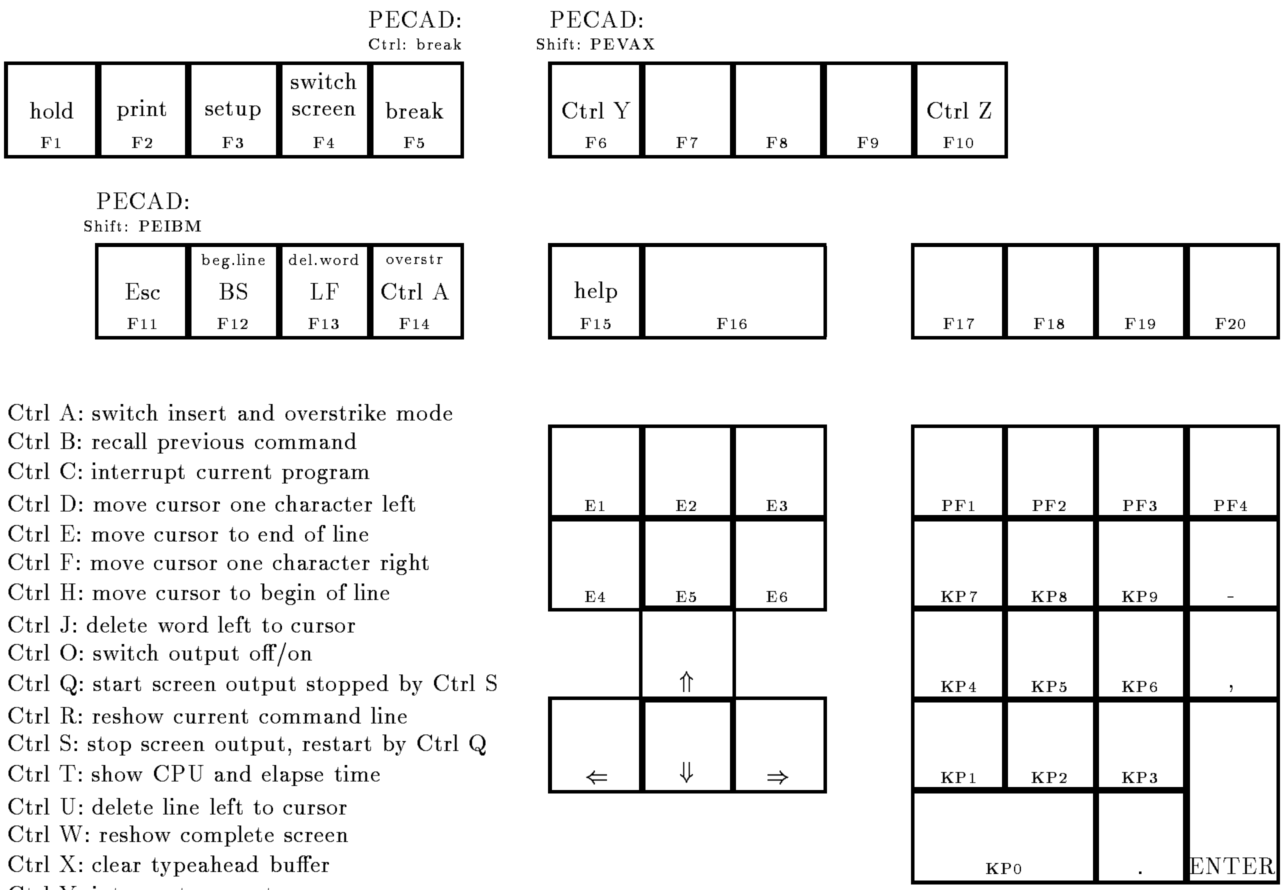
If you want to use a terminal connected to a terminal server as a graphics output device under OpenVMS (device separation) you have to get its terminal line number. To do so login at the graphics terminal to the `S200xx Local>` terminal server mode by hitting the `Return` key. Then type in the server commands

```
S200xx Local> SHOW SERVER
S200xx Local> SHOW PORT
S200xx Local> LOGOUT
```

The server information shows you the device number range `LTAxxx-LTAyyy` of that server. The `SHOW PORT` gives you the port number. Now combine them in selecting the last number of `LTAxxx` to be the port number. E.g. port number 5 and range `LTA231-LTA238` defines the terminal line number of your graphics terminal to be `LTA235:`. The `LOGOUT` command frees the port for remote access from the Alpha AXP or VAX. Now you can access this terminal from another Alpha AXP or VAX terminal using this line number, e.g.

```
$ ALLOCATE LTA235:             ! reserve terminal line for your own usage
$ COPY file.type LTA235:       ! write something on the terminal
$ DEALLOCATE LTA235:           ! free the terminal line later
```

In the figure A.1 on page 384 you see the OpenVMS DCL keypad layout for line editing.



Ctrl A: switch insert and overstrike mode
Ctrl B: recall previous command
Ctrl C: interrupt current program
Ctrl D: move cursor one character left
Ctrl E: move cursor to end of line
Ctrl F: move cursor one character right
Ctrl H: move cursor to begin of line
Ctrl J: delete word left to cursor
Ctrl O: switch output off/on
Ctrl Q: start screen output stopped by Ctrl S
Ctrl R: reshow current command line
Ctrl S: stop screen output, restart by Ctrl Q
Ctrl T: show CPU and elapse time
Ctrl U: delete line left to cursor
Ctrl W: reshow complete screen
Ctrl X: clear typeahead buffer
Ctrl Y: interrupt current program
Ctrl Z: exit current program, end of input

Figure A.1: The Special Keypad Layout for DCL Level (Line Editing).

Appendix B

Login Command Procedure

A user's login command procedure will be executed at the login time of the user. The user might define logical names or DCL symbols in this login procedure. The file must have the name `LOGIN.COM` and it must be located in the so called login directory of the user where the default is set to after the login. Normally this directory has the same name as the username. You can get this name by the translation of the logical name `SYS$LOGIN`, using the command:

```
$ SHOW LOGICAL SYS$LOGIN
   or just
$ SLOG SYS$LOGIN
```

The following listing is an example of such a login command procedure file. It is located under `GOO$EXE:USER_LOGIN.COM`. A new user might copy this file to his own login directory and edit it to his own needs.

```
$ SET NOVERIFY
$ SET NOON
$! Define names for text library:
$ DEFINE/JOB PLI$LIBRARY SYS$LOGIN:privlib.TLB
$ DEFINE/JOB tpriv SYS$LOGIN:privlib.TLB
$ DEFINE/JOB LNK$LIBRARY SYS$LOGIN:privlib.OLB
$ DEFINE/JOB opriv SYS$LOGIN:privlib.OLB
$ DEFINE/JOB HLP$LIBRARY SYS$LOGIN:privlib.HLB
$ DEFINE/JOB hpriv SYS$LOGIN:privlib.HLB
$!
$! Set DCL function keys (Press PF2 for help):
$ PFKEY
$!
$! Set system prompt to "node:user$ ":
$ @GOO$EXE:SETPROMPT.COM
$!
```

```
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$! The following lines are for all Utilities writen at GSI !!!!!
$!
$ TOOLLOGIN
$!
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$! The following lines are for Public Software, like GNU and WWW !!!!!
$!
$ PUBLICLOGIN
$!
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$! The following lines are for Software written by CERN, e.g. PAW  !!!!!
$! the parameter might be PRO, NEW or OLD depending on the version !!!!!
$!
$ CERNLOGIN NEW
$!
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$! The following lines are for the TeX and LaTeX software !!!!!
$!
$ NEWTEX
$ NEWLATEX
$!
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$! The following lines are for GOOSY users, only !!!!!
$!
$!  Define your global goosytable >LNM$GOOSY<
$!  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$   set message/nofac/nosev/notext/noident
$   create/name_table LNM$GOOSY
$   set message/fac/sev/text/ident
$!  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$!
$! Define names for profiles:
$ DEFINE/JOB GOO$PROFILE GOO$EXE:PROFILE.PROF
$ DEFINE/JOB GOO$INI_ALL GOO$EXE:INI_ALL.COM
$ DEFINE/JOB GOO$INI_TPO GOO$EXE:INI_TPO.COM
$!
$! Define all GOOSY stuff:
$ @GOO$EXE:GOOLOG.COM
$! Establish a data base for analysis control:
$ GOOCONTROL
$! Set GOOSY message output to readable format:
```

```
$ SETMES GOOSY /NOHEAD/NOPREF
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$!
$! Add here user specific statements:
$!
$ IF F$MODE() .NES. "INTERACTIVE" THEN GOTO G_BATCH
$!
$! Add here statements to execute interactively only:
$!
$ GOTO G_FINISH
$!
$ G_BATCH:
$ IF F$MODE() .NES. "BATCH" THEN EXIT
$ WRITE SYS$OUTPUT "*********** Starting user batch procedure *****************"
$!
$! Add here statements to execute in batch only:
$!
$ SET VERIFY
$ G_FINISH:
$ EXIT
```

Appendix C

GOOSY Command Keywords Summary

GOOSY_keywords

Keywords In the following the GOOSY command keywords are listed with their occurance in the commands.

$

$ CLOSE ETHERNET
$ COMMENT
$ DCL
$ DEBUG
$ DEFINE KEY
$ RECALL
$ REPEAT
$ RESET DEFAULT
$ SET DEFAULT
$ SET GNA ETHERNET
$ SHOW COMMAND
$ SHOW GNA COMPONENTS
$ SHOW GNA ETHERNET
$ SHOW GNA LINKS
$ SHOW GNA MCBS
$ SHOW GNA PROCESS
$ SHOW GNA RPC
$ SHOW GNA STATUS
$ SHOW KEY
$ SHOW MEMORY
$ SHOW PROCESS
$ SHOW TIMER

1802

TEST BOR 1802

2047

TEST SEN 2047

2090

TEST SEN 2090

2228

TEST LRS 2228

2249

TEST LRS 2249

2365

LOAD LRS 2365
STORE LRS 2365

2551

TEST LRS 2551

4432

TEST LRS 4432

4434

TEST LRS 4434

ACQUISITION

FOREIGN ACQUISITION
INITIALIZE ACQUISITION
LOAD MODULE ACQUISITION
RESET ACQUISITION
SET ACQUISITION
SHOW ACQUISITION
START ACQUISITION
STOP ACQUISITION

ALIAS

CREATE ALIAS
DELETE ALIAS
SHOW ALIAS

ALLOCATE

ALLOCATE DEVICE

ANALYSIS

ATTACH ANALYSIS
DETACH ANALYSIS
INITIALIZE ANALYSIS
LOAD MODULE ANALYSIS
SET ANALYSIS
SHOW ANALYSIS
START ANALYSIS OUTPUT
START ANALYSIS RANDOM
STOP ANALYSIS OUTPUT
STOP ANALYSIS RANDOM

AREA

CREATE AREA
SHOW AREA

ATTACH

ATTACH ANALYSIS
ATTACH BASE
ATTACH DYNAMIC LIST

ATTACHED

SHOW DYNAMIC ATTACHED

BASE

ATTACH BASE
COMPRESS BASE
CONVERT BASE
COPY BASE
CREATE BASE
DECOMPRESS BASE
DETACH BASE
DETACH BASE
DISMOUNT BASE
LOCATE BASE
MOUNT BASE
UPDATE BASE

BIT

TEST MPI BIT

BITSPECTRUM

CREATE DYNAMIC ENTRY BITSPECTRUM

BLOCK

SHOW HOME BLOCK

BOR

TEST BOR 1802

BUFFER

SET SCATTER BUFFER
SET VME BUFFER
SHOW BUFFER DUMP
SHOW SCATTER BUFFER
START BUFFER DUMP
STOP BUFFER DUMP
TYPE BUFFER

CALCULATE

CALCULATE FASTBUS PEDESTAL
CALCULATE SPECTRUM

CALIBRATE

CALIBRATE SPECTRUM

CALIBRATION

CREATE CALIBRATION FIXED
CREATE CALIBRATION FLOAT
CREATE CALIBRATION LINEAR
DELETE CALIBRATION
DISPLAY CALIBRATION
SET CALIBRATION FIXED
SET CALIBRATION FLOAT
SET CALIBRATION LINEAR
SHOW CALIBRATION

CAMAC

CAMAC CLEAR
CAMAC CNAF
CAMAC DEMAND
CAMAC INHIBIT

CAMAC INITIALIZE
CAMAC SCAN
CLEAR CAMAC SPECTRUM
INITIALIZE CAMAC
READ CAMAC SPECTRUM
RESET CAMAC
SHOW CAMAC SPECTRUM
TEST CAMAC
WRITE CAMAC SPECTRUM

CHANNEL

RELEASE MBD CHANNEL

CLEAR

CAMAC CLEAR
CLEAR CAMAC SPECTRUM
CLEAR CONDITION COUNTER
CLEAR DEVICE
CLEAR ELEMENT
CLEAR PICTURE
CLEAR SPECTRUM

CLOSE

$ CLOSE ETHERNET
CLOSE FILE
CLOSE OUTPUT FILE

CNAF

CAMAC CNAF
CNAF VME

COLOR

SET DEVICE COLOR

COMMAND

$ SHOW COMMAND

COMMENT

$ COMMENT

COMPONENTS

$ SHOW GNA COMPONENTS

COMPOSED

CREATE CONDITION COMPOSED
CREATE DYNAMIC ENTRY COMPOSED

COMPRESS

COMPRESS BASE

CONDITION

CLEAR CONDITION COUNTER
COPY CONDITION
CREATE CONDITION COMPOSED
CREATE CONDITION FUNCTION
CREATE CONDITION MULTIWINDOW
CREATE CONDITION PATTERN
CREATE CONDITION POLYGON
CREATE CONDITION WINDOW
CREATE TABLE CONDITION
DELETE CONDITION
DISPLAY CONDITION
FREEZE CONDITION
MODIFY TABLE CONDITION

REPLACE CONDITION WINDOW
SET CONDITION PATTERN
SET CONDITION WINDOW
SHOW CONDITION
UNFREEZE CONDITION

CONTROL

SET VME CONTROL
SHOW VME CONTROL

CONVERT

CONVERT BASE

COPY

COPY BASE
COPY CONDITION
COPY ELEMENT
COPY FILE
COPY MEMBER
COPY POLYGON
COPY SPECTRUM

COUNTER

CLEAR CONDITION COUNTER

CREATE

CREATE ALIAS
CREATE AREA
CREATE BASE
CREATE CALIBRATION FIXED
CREATE CALIBRATION FLOAT
CREATE CALIBRATION LINEAR
CREATE CONDITION COMPOSED
CREATE CONDITION FUNCTION

CREATE CONDITION MULTIWINDOW
CREATE CONDITION PATTERN
CREATE CONDITION POLYGON
CREATE CONDITION WINDOW
CREATE DIRECTORY
CREATE DYNAMIC ENTRY BITSPECTRUM
CREATE DYNAMIC ENTRY COMPOSED
CREATE DYNAMIC ENTRY FUNCTION
CREATE DYNAMIC ENTRY INDEXEDSPECTRUM
CREATE DYNAMIC ENTRY MULTIWINDOW
CREATE DYNAMIC ENTRY PATTERN
CREATE DYNAMIC ENTRY POLYGON
CREATE DYNAMIC ENTRY PROCEDURE
CREATE DYNAMIC ENTRY SCATTER
CREATE DYNAMIC ENTRY SPECTRUM
CREATE DYNAMIC ENTRY WINDOW
CREATE DYNAMIC LIST
CREATE ELEMENT
CREATE ENVIRONMENT
CREATE LINK
CREATE OVERLAY
CREATE PICTURE
CREATE POLYGON
CREATE POOL
CREATE PROCESS
CREATE PROGRAM
CREATE SESSION
CREATE SPECTRUM
CREATE TABLE CONDITION
CREATE TABLE SPECTRUM
CREATE TYPE

DATA

SEND DATA

DCL

$ DCL

DEALLOCATE

DEALLOCATE DEVICE

DEBUG

$ DEBUG
DEBUG VME MEMORY

DECALIBRATE

DECALIBRATE SPECTRUM

DECOMPRESS

DECOMPRESS BASE

DEFAULT

$ RESET DEFAULT
$ SET DEFAULT

DEFINE

$ DEFINE KEY
DEFINE DISPLAY HEADER
DEFINE DISPLAY PICTURE
DEFINE DISPLAY SPECTRUM
DEFINE FRAME SETUP
DEFINE PICTURE SETUP

DELETE

DELETE ALIAS
DELETE CALIBRATION
DELETE CONDITION

DELETE DYNAMIC ENTRY
DELETE DYNAMIC LIST
DELETE ELEMENT
DELETE ENVIRONMENT
DELETE LINK
DELETE OVERLAY
DELETE PICTURE
DELETE POLYGON
DELETE POOL
DELETE PROCESS
DELETE SECTION
DELETE SPECTRUM

DEMAND

CAMAC DEMAND

DETACH

DETACH ANALYSIS
DETACH BASE
DETACH BASE
DETACH DISPLAY
DETACH DYNAMIC LIST

DEVICE

ALLOCATE DEVICE
CLEAR DEVICE
DEALLOCATE DEVICE
SET DEVICE COLOR

DEVICES

SHOW DEVICES

DIRECTORY

CREATE DIRECTORY

LOCATE DIRECTORY
MODIFY DIRECTORY
SHOW DIRECTORY

DISMOUNT

DISMOUNT BASE
DISMOUNT TAPE

DISPLAY

DEFINE DISPLAY HEADER
DEFINE DISPLAY PICTURE
DEFINE DISPLAY SPECTRUM
DETACH DISPLAY
DISPLAY CALIBRATION
DISPLAY CONDITION
DISPLAY GRAPH
DISPLAY METAFILE
DISPLAY PICTURE
DISPLAY POINT
DISPLAY POLYGON
DISPLAY SCATTER
DISPLAY SPECTRUM
DISPLAY TEXT
SAVE DISPLAY
SET DISPLAY MODE
SHOW DISPLAY GLOBALS

DUMP

DUMP MBD
DUMP SPECTRUM
DUMP STARBURST
SHOW BUFFER DUMP
START BUFFER DUMP
STOP BUFFER DUMP

DYNAMIC

ATTACH DYNAMIC LIST
CREATE DYNAMIC ENTRY BITSPECTRUM
CREATE DYNAMIC ENTRY COMPOSED
CREATE DYNAMIC ENTRY FUNCTION
CREATE DYNAMIC ENTRY INDEXEDSPECTRUM
CREATE DYNAMIC ENTRY MULTIWINDOW
CREATE DYNAMIC ENTRY PATTERN
CREATE DYNAMIC ENTRY POLYGON
CREATE DYNAMIC ENTRY PROCEDURE
CREATE DYNAMIC ENTRY SCATTER
CREATE DYNAMIC ENTRY SPECTRUM
CREATE DYNAMIC ENTRY WINDOW
CREATE DYNAMIC LIST
DELETE DYNAMIC ENTRY
DELETE DYNAMIC LIST
DETACH DYNAMIC LIST
SET DYNAMIC LIST
SHOW DYNAMIC ATTACHED
SHOW DYNAMIC LIST
START DYNAMIC LIST
STOP DYNAMIC LIST
UPDATE DYNAMIC LIST

ELEMENT

CLEAR ELEMENT
COPY ELEMENT
CREATE ELEMENT
DELETE ELEMENT
LOCATE ELEMENT
SHOW ELEMENT

ENTRY

CREATE DYNAMIC ENTRY BITSPECTRUM
CREATE DYNAMIC ENTRY COMPOSED
CREATE DYNAMIC ENTRY FUNCTION
CREATE DYNAMIC ENTRY INDEXEDSPECTRUM
CREATE DYNAMIC ENTRY MULTIWINDOW
CREATE DYNAMIC ENTRY PATTERN
CREATE DYNAMIC ENTRY POLYGON

CREATE DYNAMIC ENTRY PROCEDURE
CREATE DYNAMIC ENTRY SCATTER
CREATE DYNAMIC ENTRY SPECTRUM
CREATE DYNAMIC ENTRY WINDOW
DELETE DYNAMIC ENTRY

ENVIRONMENT

CREATE ENVIRONMENT
DELETE ENVIRONMENT

ETHERNET

$ CLOSE ETHERNET
$ SET GNA ETHERNET
$ SHOW GNA ETHERNET

EVENT

SET EVENT INPUT
SET EVENT OUTPUT
TYPE EVENT

EXECUTE

EXECUTE VME

EXPAND

EXPAND

FASTBUS

CALCULATE FASTBUS PEDESTAL
SET FASTBUS PEDESTAL

FILE

CLOSE FILE
CLOSE OUTPUT FILE
COPY FILE
OPEN FILE
OPEN OUTPUT FILE
START INPUT FILE
START OUTPUT FILE
STOP INPUT FILE
STOP OUTPUT FILE
TYPE FILE

FIT

FIT SPECTRUM

FIXED

CREATE CALIBRATION FIXED
SET CALIBRATION FIXED

FLOAT

CREATE CALIBRATION FLOAT
SET CALIBRATION FLOAT

FOREIGN

FOREIGN ACQUISITION

FRAME

DEFINE FRAME SETUP
MODIFY FRAME SCATTER
MODIFY FRAME SPECTRUM
ZOOM FRAME

FRAMES

UPDATE FRAMES

FREEZE

FREEZE CONDITION
FREEZE SPECTRUM

FUNCTION

CREATE CONDITION FUNCTION
CREATE DYNAMIC ENTRY FUNCTION

GLOBALS

SHOW DISPLAY GLOBALS

GNA

$ SET GNA ETHERNET
$ SHOW GNA COMPONENTS
$ SHOW GNA ETHERNET
$ SHOW GNA LINKS
$ SHOW GNA MCBS
$ SHOW GNA PROCESS
$ SHOW GNA RPC
$ SHOW GNA STATUS

GOOSY

SHOW GOOSY STATUS

GRAPH

DISPLAY GRAPH

GSI

TEST GSI IOL

HEADER

DEFINE DISPLAY HEADER

HOME

SHOW HOME BLOCK

HVR

ID

LOCATE ID

INDEXEDSPECTRUM

CREATE DYNAMIC ENTRY INDEXEDSPECTRUM

INHIBIT

CAMAC INHIBIT

INITIALIZE

CAMAC INITIALIZE
INITIALIZE ACQUISITION
INITIALIZE ANALYSIS
INITIALIZE CAMAC

INPUT

SET EVENT INPUT
SET VME INPUT
START INPUT FILE
START INPUT MAILBOX

START INPUT NET
STOP INPUT FILE
STOP INPUT MAILBOX
STOP INPUT NET

INTEGRATE

INTEGRATE

IOL

TEST GSI IOL

J11

LOAD J11

KEY

$ DEFINE KEY
$ SHOW KEY

KEYPAD

SHOW TP0 KEYPAD

LETTERING

SET LETTERING

LINEAR

CREATE CALIBRATION LINEAR
SET CALIBRATION LINEAR

LINK

CREATE LINK
DELETE LINK
SHOW LINK

LINKS

$ SHOW GNA LINKS

LIST

ATTACH DYNAMIC LIST
CREATE DYNAMIC LIST
DELETE DYNAMIC LIST
DETACH DYNAMIC LIST
SET DYNAMIC LIST
SHOW DYNAMIC LIST
START DYNAMIC LIST
STOP DYNAMIC LIST
UPDATE DYNAMIC LIST

LOAD

LOAD J11
LOAD LRS 2365
LOAD MBD
LOAD MODULE ACQUISITION
LOAD MODULE ANALYSIS
LOAD STARBURST
LOAD VME PROGRAM
LOAD VME TABLE

LOCATE

LOCATE BASE
LOCATE DIRECTORY
LOCATE ELEMENT
LOCATE ID
LOCATE POOL
LOCATE QUEUEELEMENT

LOCATE TYPE

LOCK

SET LOCK OUTPUT

LOCKS

SHOW LOCKS

LRS

LOAD LRS 2365
STORE LRS 2365
TEST LRS 2228
TEST LRS 2249
TEST LRS 2551
TEST LRS 4432
TEST LRS 4434

MAILBOX

START INPUT MAILBOX
STOP INPUT MAILBOX

MAPPING

SHOW MAPPING
SHOW MAPPING
SHOW MAPPING

MBD

DUMP MBD
LOAD MBD
PATCH MBD
RELEASE MBD CHANNEL
RESET MBD

STORE MBD

MCBS

$ SHOW GNA MCBS

MEMBER

COPY MEMBER
SET MEMBER
SHOW MEMBER

MEMORY

$ SHOW MEMORY
DEBUG VME MEMORY

MESSAGE

METAFILE

DISPLAY METAFILE
PLOT METAFILE

MODE

SET DISPLAY MODE

MODIFY

MODIFY DIRECTORY
MODIFY FRAME SCATTER
MODIFY FRAME SPECTRUM
MODIFY TABLE CONDITION
MODIFY TABLE SPECTRUM

MODULE

LOAD MODULE ACQUISITION
LOAD MODULE ANALYSIS

MOUNT

MOUNT BASE
MOUNT TAPE

MPI

TEST MPI BIT
TEST MPI TDC

MR2000

START MR2000
STOP MR2000

MULTIWINDOW

CREATE CONDITION MULTIWINDOW
CREATE DYNAMIC ENTRY MULTIWINDOW

MWPC

SET MWPC

NET

START INPUT NET
STOP INPUT NET

OPEN

OPEN FILE
OPEN OUTPUT FILE

OUTPUT

CLOSE OUTPUT FILE
OPEN OUTPUT FILE
SET EVENT OUTPUT
SET LOCK OUTPUT
START ANALYSIS OUTPUT
START OUTPUT FILE
STOP ANALYSIS OUTPUT
STOP OUTPUT FILE

OVERLAY

CREATE OVERLAY
DELETE OVERLAY
OVERLAY

PATCH

PATCH MBD
PATCH STARBURST

PATTERN

CREATE CONDITION PATTERN
CREATE DYNAMIC ENTRY PATTERN
SET CONDITION PATTERN

PEDESTAL

CALCULATE FASTBUS PEDESTAL
SET FASTBUS PEDESTAL

PICTURE

CLEAR PICTURE

CREATE PICTURE
DEFINE DISPLAY PICTURE
DEFINE PICTURE SETUP
DELETE PICTURE
DISPLAY PICTURE
PLOT PICTURE
SHOW PICTURE

PLOT

PLOT METAFILE
PLOT PICTURE
PLOT PLOTFILE

PLOTFILE

PLOT PLOTFILE

POINT

DISPLAY POINT
SET SPECTRUM POINT

POLYGON

COPY POLYGON
CREATE CONDITION POLYGON
CREATE DYNAMIC ENTRY POLYGON
CREATE POLYGON
DELETE POLYGON
DISPLAY POLYGON
REPLACE POLYGON
SHOW POLYGON

POOL

CREATE POOL
DELETE POOL
LOCATE POOL

SHOW POOL

PRINT

PRINT

PROCEDURE

CREATE DYNAMIC ENTRY PROCEDURE

PROCESS

$ SHOW GNA PROCESS
$ SHOW PROCESS
CREATE PROCESS
DELETE PROCESS

PROGRAM

CREATE PROGRAM
LOAD VME PROGRAM

PROJECT

PROJECT

PROTECT

PROTECT SPECTRUM

PROTOCOL

PROTOCOL

QUEUEELEMENT

LOCATE QUEUEELEMENT

RANDOM

SET RANDOM
START ANALYSIS RANDOM
STOP ANALYSIS RANDOM

READ

READ CAMAC SPECTRUM

RECALL

$ RECALL

REFRESH

REFRESH

REGISTER

TEST REGISTER

RELEASE

RELEASE MBD CHANNEL

REPEAT

$ REPEAT

REPLACE

REPLACE CONDITION WINDOW
REPLACE POLYGON

RESET

$ RESET DEFAULT
RESET ACQUISITION
RESET CAMAC
RESET MBD

RPC

$ SHOW GNA RPC

RUN

START RUN
STOP RUN

SAVE

SAVE DISPLAY

SCAN

CAMAC SCAN

SCATTER

CREATE DYNAMIC ENTRY SCATTER
DISPLAY SCATTER
MODIFY FRAME SCATTER
SET SCATTER BUFFER
SHOW SCATTER BUFFER
START SCATTER
STOP SCATTER

SECTION

DELETE SECTION

SEN

TEST SEN 2047
TEST SEN 2090

SEND

SEND DATA

SESSION

CREATE SESSION

SET

$ SET DEFAULT
$ SET GNA ETHERNET
SET ACQUISITION
SET ANALYSIS
SET CALIBRATION FIXED
SET CALIBRATION FLOAT
SET CALIBRATION LINEAR
SET CONDITION PATTERN
SET CONDITION WINDOW
SET DEVICE COLOR
SET DISPLAY MODE
SET DYNAMIC LIST
SET EVENT INPUT
SET EVENT OUTPUT
SET FASTBUS PEDESTAL
SET LETTERING
SET LOCK OUTPUT
SET MEMBER
SET MWPC
SET RANDOM
SET SCATTER BUFFER
SET SPECTRUM POINT

SET VME BUFFER
SET VME CONTROL
SET VME INPUT
SET VME TRIGGER

SETUP

DEFINE FRAME SETUP
DEFINE PICTURE SETUP
SHOW VME SETUP

SHOW

$ SHOW COMMAND
$ SHOW GNA COMPONENTS
$ SHOW GNA ETHERNET
$ SHOW GNA LINKS
$ SHOW GNA MCBS
$ SHOW GNA PROCESS
$ SHOW GNA RPC
$ SHOW GNA STATUS
$ SHOW KEY
$ SHOW MEMORY
$ SHOW PROCESS
$ SHOW TIMER
SHOW ACQUISITION
SHOW ALIAS
SHOW ANALYSIS
SHOW AREA
SHOW BUFFER DUMP
SHOW CALIBRATION
SHOW CAMAC SPECTRUM
SHOW CONDITION
SHOW DEVICES
SHOW DIRECTORY
SHOW DISPLAY GLOBALS
SHOW DYNAMIC ATTACHED
SHOW DYNAMIC LIST
SHOW ELEMENT
SHOW GOOSY STATUS
SHOW HOME BLOCK

SHOW LINK
SHOW LOCKS
SHOW MAPPING
SHOW MAPPING
SHOW MAPPING
SHOW MEMBER
SHOW PICTURE
SHOW POLYGON
SHOW POOL
SHOW SCATTER BUFFER
SHOW SPECTRUM
SHOW STARBURST
SHOW TABLE
SHOW TP0 KEYPAD
SHOW TREE
SHOW TYPE
SHOW VME CONTROL
SHOW VME SETUP

SLEEP

SLEEP

SPECTRUM

CALCULATE SPECTRUM
CALIBRATE SPECTRUM
CLEAR CAMAC SPECTRUM
CLEAR SPECTRUM
COPY SPECTRUM
CREATE DYNAMIC ENTRY SPECTRUM
CREATE SPECTRUM
CREATE TABLE SPECTRUM
DECALIBRATE SPECTRUM
DEFINE DISPLAY SPECTRUM
DELETE SPECTRUM
DISPLAY SPECTRUM
DUMP SPECTRUM
FIT SPECTRUM
FREEZE SPECTRUM
MODIFY FRAME SPECTRUM

MODIFY TABLE SPECTRUM
PROTECT SPECTRUM
READ CAMAC SPECTRUM
SET SPECTRUM POINT
SHOW CAMAC SPECTRUM
SHOW SPECTRUM
SUMUP SPECTRUM
UNFREEZE SPECTRUM
UNPROTECT SPECTRUM
WRITE CAMAC SPECTRUM

STARBURST

DUMP STARBURST
LOAD STARBURST
PATCH STARBURST
SHOW STARBURST

START

START ACQUISITION
START ANALYSIS OUTPUT
START ANALYSIS RANDOM
START BUFFER DUMP
START DYNAMIC LIST
START INPUT FILE
START INPUT MAILBOX
START INPUT NET
START MR2000
START OUTPUT FILE
START RUN
START SCATTER
START VME

STATUS

$ SHOW GNA STATUS
SHOW GOOSY STATUS

STOP

STOP ACQUISITION
STOP ANALYSIS OUTPUT
STOP ANALYSIS RANDOM
STOP BUFFER DUMP
STOP DYNAMIC LIST
STOP INPUT FILE
STOP INPUT MAILBOX
STOP INPUT NET
STOP MR2000
STOP OUTPUT FILE
STOP RUN
STOP SCATTER
STOP VME

STORE

STORE LRS 2365
STORE MBD

SUMUP

SUMUP SPECTRUM

TABLE

CREATE TABLE CONDITION
CREATE TABLE SPECTRUM
LOAD VME TABLE
MODIFY TABLE CONDITION
MODIFY TABLE SPECTRUM
SHOW TABLE

TAPE

DISMOUNT TAPE

MOUNT TAPE

TDC

TEST MPI TDC

TEST

TEST BOR 1802
TEST CAMAC
TEST GSI IOL
TEST LRS 2228
TEST LRS 2249
TEST LRS 2551
TEST LRS 4432
TEST LRS 4434
TEST MPI BIT
TEST MPI TDC
TEST REGISTER
TEST SEN 2047
TEST SEN 2090

TEXT

DISPLAY TEXT

TIMER

$ SHOW TIMER

TP0

SHOW TP0 KEYPAD

TREE

SHOW TREE

TRIGGER

SET VME TRIGGER

TYPE

CREATE TYPE
LOCATE TYPE
SHOW TYPE
TYPE BUFFER
TYPE EVENT
TYPE FILE

UNFREEZE

UNFREEZE CONDITION
UNFREEZE SPECTRUM

UNPROTECT

UNPROTECT SPECTRUM

UPDATE

UPDATE BASE
UPDATE DYNAMIC LIST
UPDATE FRAMES

VME

CNAF VME
DEBUG VME MEMORY
EXECUTE VME
LOAD VME PROGRAM
LOAD VME TABLE
SET VME BUFFER

SET VME CONTROL
SET VME INPUT
SET VME TRIGGER
SHOW VME CONTROL
SHOW VME SETUP
START VME
STOP VME

VOICE

WAIT

WAIT

WINDOW

CREATE CONDITION WINDOW
CREATE DYNAMIC ENTRY WINDOW
REPLACE CONDITION WINDOW
SET CONDITION WINDOW

WRITE

WRITE CAMAC SPECTRUM

ZOOM

ZOOM FRAME

Appendix D

DCL Commands Summary

acctng	@accttng p1 p2 p3 Takes the ACCOUNTNG file of VMS and do selective inquiries using ACCOUNTING (the VMS utility).
ALIAS	command arguments Handle GOOSY alias command names
ATENVIR	ATENV*IR environment Attach environment.
BFXT	@GOO$EXE:BFXT inirep action file /ID=MYDATA/MODE=CHAR /FILELN=fileln/MSGLEVEL=msglevel/NOEXECUTE send data to the IBM via BFX/NETEX
CADDRESS	infile outfile /PRINTER=p Create and optional print address files..
CADJUST	infile outfile /ADJUST/TAB Adjust GOOSY documentation headers.
CALLEX	library filespec GL-switches Extract Calling sequence from PLI source.
CALLTREE	input /MODULE= /OUTPUT= /TT /EXCLUDE= Makes a calling tree out of a cross reference
CBACKUP	P1 P2 ... P8 Execute the VMS BACKUP utility and return an exit status based on the analysis of all errors.
CBATCHDCL	"dcl_line1" "dcl_line2" "dcl_line3" "dcl_line4"

	/LOG_FILE=file /DEFAULT=dir /QUEUE=q /AFTER=time /NAME=n /CHARACTERISTICS=c/CPUTIME=t/RESTART /NODE=n /VERIFY/ACCOUNTING /WARNING/ERROR/FATAL Executes up to 4 DCL command lines in a BATCH job under the current default directory either on the local or an remote node.
CBINHEX	in_file out_file This procedure creates a HEX file containing the FDL description and the contents in hexadecimal format of another file. Those HEX files can be converted back with CHEXBIN and are usefull for sending them over stupid communication links.
CDBLQUOTE	input Replace every quote in a text string by a double quote.
CDCLCOM	inc module performs a compile of PL/I text contained in 'file' within a dummy frame (GOO$DM:U$DCLDU.PPL)
CDCLLIST	dsc p1 p2 p3 p4 p5 p6 p7 This procedure provides a mechanism which allows to pass parameters and qualifiers to DCL procedures in the same way as to DCL commands.
CDELSYM	prefix start end Delete global symbols created by CNAMELIST or CWILDCARD
CDIFFER	file1 file2 /LIST/PARA/DIFFER Allows wildcarded DIFFERENCE with suppressed output.
CDIRO	file-spec. /SIZE /DATE /FULL/TEST sorts a DIR command output
CDOCUM	CDOC module type formatter library Generates documentation from source file and inserts text module to library
CEASYMAIL	@GOO$EXE:CEASYMAIL file recipient /SUBJECT=s/DELETE Send a file as MAIL to a High Energy Physics DECnet user via EARN via the Mailer at CUNYVM.
CEDITDOC	CED file key Calls MGENHEAD for header generation and the VMS editor LSEDIT.

CFILTYPES	CFILT*YPES file switches /D*IRECT Outputs list of all file types on a directory
CHANAL	infile outfile /COPY/FULL Check GOOSY analysis programs.
CHECKDATE	newfile oldfile Checks wether newfile is younger then oldfile (status=1) or not (status=11)
CHEPMAIL	@GOO$EXE:CHEPMAIL file recipient /SUBJECT=s/DELETE Send a file as MAIL to a High Energy Physics DECnet user via EARN via the Mailer at CERNVAX.
CHEXBIN	in_file out_file This procedure processes a HEX file send of a stupid communication network and creates the file described in the HEX file.
CINCHLP	CINC*HLP format incl.libr.(module) help-libr. Generates documentation of include modules.
CINSNUM	CINS input output Inserts line numbers (9 char.) to source files
CLCOUNT	file symbol /NOHEAD Outputs number of lines of source files
CLINK	CL file/switches /switches Link programs. Defaults are updated.
CMAIL	file recipient /ED*IT /DEL*ETE /SUB*JECT=s /NOSE*LF /SPOOL=s Send or spool a MAIL with VMSmail or other mail systems on V8600B only
CMANUAL	CMAN <utility list> <format> <prefix> Composes GOOSY manuals (Programs, modules, procedures)
CMESSAGE	facility [switch] Utility to insert a new message in a message file
CMODMAP	file /COMPILE /OUTPUT=file Outputs list of modules called by module in file. (Only first level calls)
CMSGLIST	CMSG*LIST [directory][module] Generates script and help file of all GOOSY messages

CMTBACK	output Perform an image system backup for all disk devices on a magtape without verification.
CNAMELIST	<namelist> <prefix> /DIR*ECTORY /SHORT /LIB*RARY=<library> /SEL*ECT=<string> /EXC*LUDE=<string> /SEA*RCH=<string> /SIN*CE=<date> /BEF*ORE=<date> /OUT*PUT=<file> Defines a set of symbols with names got from <namelist>.
CNOTICE	recipient/NOSELF "subject" "line 1" ... "line 6" Send a short notice with VMSmail
COMPILE	COM*PILE file /PRE*QUAL=(list)/Q*UALIFIER=(list) /G*IPSY=list/LIBRARY=(list)/DEBUG/KEEP /OLB=library/SINCE=time/BEFORE=time/NEWPLI /FAST/MACRO/CALL/BATCH/COMPILE General compile procedure for all compilers THE PREVIOUS VERSION CAN BE CALLED BY OCOMPILE !
CONCAT	infile outfile /LOG/DELETE/CONFIRM/APPEND Concatenates input files to one output file.
COPTLIST	list keylist Parse IBM styled optional parameter list for DCL procedures.
CPLICOM	PPL_file/switches /switches %DEB %NEWPLI Compile PLI programs including certain text libraries. Defaults are updated.
CPRECOM	CPRE*COM file(switches) Call precompiler
CPRINT	file_list /DESTINATION=d Print a file on a spooled printer.
CPURGE	CPU*RGE filespec [purge qual.] Does a secure purging

CREDB	basename filename size[KB] /DYNLISTS=d /SPECTRA=s /CONDITIONS=c /PICTURES=p /DIRECTORIES=d /POOLS=p /POLYGONS=p /NEW /SAVE=file Create an preformat a new GOOSY data base.
CREMNUM	CREM input output col check Removes line numbers (<col> char.) from source files checking first <check> characters to be digits.
CRENVIR	CRENV*IR environment program component /ONLINE/OFFLINE/DEFAULT /$TMR/$ANL/$DSP/$DBM/J11 /[NO]DECWINDOW /PRIORITY=p/DELETE Creates a GOOSY environment and optional some GOOSY standard components
CREPEAT	CREP*EAT dcl_line Repeats a dcl command continuously, beginning on screen top.
CREPLACE	CREPL*ACE file old new /U*PPER/P*RINT/F*ORMAT/OUT=file Replaces old string by new calling MREPLACE. Documentation headers will be adjusted optionally.
CSYMDIR	@GOO$EXE:CSYMDIR Create symbols for SET DEFAULT from the directory tree
CSYMHLP	CSYM input output Calls MSYMHLP to generate help files from command procedures.
CTEXCOM	file /NOLIST/NOLATEX/PASSES=n/DVI=drv/DELETE=l 'Compile' TEX source using PLAIN TeX ot LaTeX
CTEXMANUAL	manual /REF*ORMAT /REL*EASE=r/VER*SION=v/BR*IEF Create GOOSY manual
CTRL_T	[process][output] Similar to an interactive CTRL_T, but works in DCL procedures.
CVTISOL	<input file>,<output file>[/SHORT] Convert ISOLDE spectra into SATAN readable format.
CWHAT	options arguments Activates the WHAT Utility to display details about the VMS system status.

CWV	– inquires a diary, 'Wiedervorlage'
D0_BACK	@D0_BACK output ALL Perform an image backup for disk device DUA0: on a magtape without verification.
DCFIBM	vaxfile "parm list" Sends VAXfile to IBM DCFTEMP.TEXT and starts batch job calling DCF on IBM
DLENVIR	DLENV*IR Delete current environment and all subprocesses
DOCIBM	DOC vaxfile [dest]PROP TWOPASS Sends a file to IBM DOCTEMP.TEXT(DUMMY) and starts batch job calling DOC
Document	Desciption of the module documentation tools The following commands are provided to generate documentation for VMS help utility and printer output (RNO and SCRIPT).
DTENVIR	DTENV*IR Detach environment.
DVIIBM	dvifile switches Sends DVI-file to IBM. via batch job
DVIPRI	– file(s) /DEVICE=dev /STARTPAGE=spage /PAGES=npage /DELETE /SEP*ARATE/[NO]PR*INT Print TeX's device independant (DVI) file
ECLINE	ECL*INE dcl-line /LIS*T /CONF*IRM /NOASS*IGN /$$1=list /$$2=list /1LIB*RARY=lib /2LIB*RARY=lib /1SIN*CE=date /2SIN*CE=date /1SH*ORT /2SH*ORT /1DIR*ECT /2DIR*ECT /1SEA*RCH=list /2SEA*RCH=list Execute dcl line with dummies replaced from list
EDDT	file /READ/PROFILE=prof Call full screen edit with a profile depending on the file type.

Error_Handling	Error and Message Handling, User's Guide Vers. 1.05 This documentation is available on-line, say 'HELP ERROR_HANDLING'.
ETHDEF	destination ethernet protocol interface Define logicals for ethernet connection to VME.
EXTRCALL	libfile Calls MCALLSYS or MCALLRTL to extract procedure call statments from a listing LIB /EXTRACT from a library
GIPSY	input output /LIST/DELETE Call GIPSY processor.
GLCNVPROJECT	@GOO$EXE:GLCNVPROJECT file project Convert project name within a history file
GLCNVV31	@GOO$EXE:GLCNVV31 file Convert history file format from version 3.1 to 4.0
GLCOMPILE	file Compile a file in the local test environment
GLCREATE	option arguments Creates datasets or entries in the history file directories.
GLDELETE	option arguments Deletes either datasets or information in the history file.
GLDOCUMENT	GLDOC*UMENT filespec /PUT extr.qual form.qual Generate documentation with GLEXTRACT and GLFORMAT.
GLEDIT	dataset /COMMENT=c/NOMARK/NOSUBMIT/NOCONFIRM Edit a dataset.
GLEXTRACT	file_list /MLIB=ml /CLIB=cl /MTLB=mt /MHLB=mh /CTLB=ct /CHLB=ch /TYPE=t/NOSUBMIT /SORT /DIRECT=file /CALL*LIB=tl Extract documentation from source files or libraries and store in libraries or file.
GLFORMAT	input output /SELECT=sl /EXCLUDE=el /TEX/SCRIPT/HELP/PRINT/RNO /BRIEF/COMMAND/TOC/HL2/LEV2 /LIB=l /HLB=ml /NOSUBMIT /LOG/DIAGNOSIS Format extracted documentation headers.

GLGET	dataset file /COMMENT="comm" Copy a dataset from a project directory or library to the current working directory.
GLINFO	key Call GOOSY information system
GLMAIL	"subject" /AUTHOR/USER/PROJECT/KEY=key/CONF=conf Send MAIL to current projects author group or user group and add this message in the appropriate news file.
GLMANAGER	option arguments Performs all project manager functions.
GLPUT	file dataset /COMMENT="comm" /SOURCE/GENERATED/FOREIGN /NOMARK /NOCORRELATE /NOSUBMIT /DOCUMENT /COMPILE /LINK /PROLOGUE /EPILOGUE /UPDATE /NEW/NOCONFIRM/NODELETE /MFORMAT=lib/CFORMAT=lib Store a file from the working directory to a project directory or library.
GLRELEASE	dataset /COMMENT=c /NOINTERLOCK Release a locked dataset without storing a dataset.
GLSATTR	GLSAT*TR type module /UPD*ATE Set attributes for certain types of modules
GLSET	option arguments Defines or changes attributes and characteristics of datasets in the history file or modifies execution characteristics of module management operations.
GLSHOW	option arguments Displays information about the current status of the project and it's modules.
GLTEST	option arguments Collects all operations of the local test environment.
GLTOOL	option arguments This command is a collection is software development tools available under the module managemant system.

GLUPDATE	dataset /NOMARK /NOCORRELATE /NOSUBMIT /DOCUMENT /COMPILE /LINK /PROLOGUE /EPILOGUE /UPDATE Mark a file for update actions.
GNEWS	outfile /SYSTEM/FILE Output all GOOSY mails.
GNOTES	command Read or write notes in one of the GOOSY notebooks.
GOOCONTROL	GOOC*ONTROL [CREATE]or [DISMOUNT] Defines logical name GOO$CONTROL for control data base. The data base is created and mounted optionally.
GUIDE	facility level /INIT=string/BRIEF/LIST/LASER Menu driven guide to use facilities.
HLPSCR	help-library module Extracts modules from a Help library and generates a new output file for SCRIPT
IBMSUBMIT	IBMSUBMIT vaxfilejcl /MAILADDR= /NOANSWER/MSGLEVEL= submit a JCL on the ibm and send optionally back the resulting output to the VAX
LANL	LA*NL obj_list /OLB=objlib/OPT=optfile/CMD=cmdfile /EXE=exefile /MAP=mapfile /DEBUG /SHARE/NOSHARE Link user specific analysis program
LIBCOPY	source_lib source_mod target_lib target_mod /EDIT /GL /LOG Copies text modules from one library to another.
LIBDEL	LIBD*EL library module Handle text library modules
LIBDIFF	source_lib source_mod target_lib target_mod /DIFFER Compare text modules from two libraries.

LIBEXTR	library module output Extract modules from text library.
LIBLIS	library module /FULL/SIN*CE=time /BEF*ORE=time/EXT*RACT/OUT*PUT=file-spec Lists or extracts specified modules of a library
LIBSEARCH	LIBS*EARCH library module list="search args" /SINCE=time/BEFORE=time/FILE Calls SEARCH on modules temporarily extracted from libraries (text only).
LIBTYPE	LIBT*YPE library module /PRINT /EDIT Handle text library modules
LINKJ11	objfile /COMPILE Link a J11 stand alone task
LOADKEYS	– Load F-keys of VT200/VT300.
LSHARIM	module image /GLOBAL=list /SHARE*LOG=name /MAP=mapfile /KEEP /GROUP /DEBUG Link modules into a sharable image.
MESDEF	facility /CLIB=/[NO]NEW/GLPUT/DELETE/LIST Generate message definition file for C programs.
MOVE	filespec dest /CONF /LOG Copies files and deletes them on source directory.
MTAPE	device name /INI*TIALIZE/DENS*ITY=d/BLOCK*SIZE=b/DIS*MOUNT Initialize and mount a GOOSY tape
NEWMOD	* /SINCE=<date> /HELP Outputs a list of all new help modules
NWCOPY	node source dest Copy one or more files to one or more nodes

NWDCL	node dcl_line output Execute a single DCL command line on one or more nodes.
NWDEFINE	user paszwort Define logical names for DECNET functions
NWDIFDIR	node file_spec Compare a directory on a remote node with the local directory and create a list of differences.
NWDIRECT	node dir_spec dir_qual Compares directories on different nodes
NWLIBRARY	node library file qual Perform a library operation on one or more nodes
NWUPDATE	node file_list /DESTINATION=d /EXCLUDE=l /LOG/JOURNAL /SINCE=t/BEFORE=t /MODIFIED/CREATED/EXPIRED/BACKUP /REPLACE/OVERLAY/NEW_VERSION /GENERIC Transfer a set of files to one or more nodes using the BACKUP utility.
OPSER	command Execute priviledged operator commands
PFKEY	– Define terminal auxiliary keypad keys.
PLOTMET	metafile type command plotter /COPIES=c /FONT=f Plot a metafile on specified plotter
PRIL	vaxfile switches Sends VAX-file to IBM PRILTMP1.LIST and starts batch job calling PRIL on IBM (VAX-780 only)
PRILS	file /PROP prints VAX sources on the laser printer
PROMPT	prompt-string default /REQUIRED help Prompt input from SYS$COMMAND

RIBM	ibmds vaxfile /COL=/CHECK=/MEMLIST=/$ALL/INTERACTIVE/NOANSWER Send IBM dataset to VAX file.
RRUN	RR file/switches /switches Runs programs. Defaults are updated.
SCRIBM	SCRI vaxfile [profile][edit] Sends VAX-file to IBM SCRITEMP.TEXT and starts batch job calling SCRIPT on IBM
SELECT_MBD	mbd Select a valid MBD controller on a VAX
SETMESSAGE	facility qualifier Control Message output of GOOSY and VMS
SETSYM	symbol value Checks if symbol already exist and outputs message in this case. Sets symbol to value.
SIBM	vaxfile ibmds /MODE= /MAILADDR=/NOANSWER/INTERACTIVE Sends VAX-file(s) to IBM-dataset(s) and overwrites existing datasets or members
SIBMGKS	metafile /PLOTTER=p Output GKS-Metafile on Plotter RP01,RP02
SIBMSPEC	datafile VSAMlib /RUNID=runid /TIME= /NOL*IST Dump GOOSY spectra to SATAN VSAM library
SSYMBOL	SSYM [<name>][/SEARCH=<string>] Displays symbol translation
SWSIZE	proc.-name /W*SMIN=min/S*TATUS=status/R*EPEAT=n Show processes with their workin set sizes.
TDOCUMENT	TDOC*UMENT file /TOC /INDEX /GOOSY /REP*ORT /REL*EASE=r /VER*SION=v /TITLE=tt /AU*THORS=a /LABEL=lt /LOGO=ll Format TeX source, e.g. produced by GLFORMAT of GLDOC, in the GOOSY document style

TLOCK	– Lock terminal by password
VMESTRUC	inputfile /PLI/FOR/C/PLIB=/CLIB=/FLIB= /GLPUT/DELETE Generate declarations from language independent source.
VMS_Primer	Short introduction for VAX users at GSI. "Common DEC-IBM PLI Standard" by W.F.J.Mueller and H.G.Essel "GOOSY Conventions (Standards)" by H.G.Essel
WCLOSE	file Wait for file to be closed.
MADDR	– Format addresses from the source INPUT and writes the result to the destination OUTPUT
MADJUST	input[,output]/ADJUST/TAB Adjusts right margins of documentation headers
MANALCH switches	MANALCH Check GOOSY analysis routines.
MANL	RUN GOO$EXE:MANL Attach and execute dynamic lists
MBASE	– Mount/dism GOOSY data base.
MBINHEX	binfile Convert binary files to ASCII HEX format.
MCALLEX	/PPL/PLI Extract calling sequence.
MCALLRTL	– Extracts call-statements out of a GOORTL listing
MCALLSYS	– Extracts calling-statements out of a listing from PLISTRARLET with the system services
MCALLTREE	[module-name] Makes a calling tree out of a cross reference list

MCMD	R GOO$EXE:MCMD Test and demonstration program for the command dispatcher. This programm is used to validate command dispatcher functions after changes and is part of the command dispatcher tuturial
MCOMHLP	input output [/HELP] Reformats a text module generated by MEXTHEA for command description for MFORMDO.
MCTRL	STS=MCTRL Control inactive users
MDBCOPY	– Compress and decompress GOOSY data bases.
MDBM	– Activate different Data Base Management Activities.
MDCLANAL	MDCL*ANAL <command line> Commands to analyze DCL procedures (Call tree) and generate 'debug' versions.
MDCLDEB	MDCLD <key list> Generates debug versions of command procedures.
MDCLLIST	"dsc" p2 p3 .. p8 This procedure provides a mechanism which allows to pass parameters and qualifiers to DCL procedures in the same way as to DCL commands.
MDISP	or by display commands GOOSY display program
MDVICNVV	RUN GOO$EXE:MDVICNV Convert TeX DVI-file to ASCII Hex-code
MEXTHEA	MEXT Extracts documentation blocks generated by MGENHEAD and generates a text module for MFORMDOC
MFIC_CTRL	– Activate different Activities using the command dispatcher.
MFORMDO	MFORM list Generates SCRIPT and RUNOFF files from textmodules created by MEXTHEA for help and documentation

MGENCIM	– Generate card image file to send to IBM
MGENHEA	MGEN language type mode Generates interactively documentation headers for programs, procedures and command procedures.
MGNS_ESONE	RUN GOO$EXE:MGNS_ESONE Network object to perform remote CAMAC accesses.
MGOOWAIT	process Wait for analysis completion
MGTOOL	GTOOL [LIBRARY—DOCUMENT—MESSAGE] Calls GOOSY tools like code management or documentation tools by menu.
MGUIDE_DISP	RUN MGUIDE_DISP Called in guide.com to display one menu
MHEXBIN	binfile Convert ASCII HEX format file created by BINHEX to binary files.
MHLPSCR	from DCL Procedure HLPSCR Generates a file for SCRIPT output from a Help file
MHLPTEX	from DCL Procedure HLPTEX Generates a file for TEX output from a Help file
MINSNUM	MINS Inserts line numbers to PLI source files
MJCLTRIM	file trimmedfile trim JCL output files from the IBM
MLCOUNT	input /LIST /NOHEAD /PPL/PLI/FOR/MAR/COM Looks for longest record and assignes value to DCL symbols MLCOUNT_MAX_LENGTH and MLCOUNT_LINES.
MLIBWILD	<module spec>[/FULL] Reads input as generated by LIB/LIST and outputs list of matching module names.
MLOCKS	– Show VMS locks.

MMESDEF	<facility> Generate PL1 program to generate file for messages linked.
MMESLIST	MMES <facility> List all messages linked.
MMODMAP	switches Formats output from "SEARCH module.lis ENTRY"
MOPER	– Execute privileged operator commands.
MPFKEY	or <PF2> Display of DCL auxiliary keypad definition
MPLOTMET	GOOEXE:MPLOTMET metafile,type,command, queue,copies,font Send a metafile to a plotter.
MPOSTRIBM	RUN GOO$EXE:MPOSTRIBM file columns removes leading numbers, performs necessary character conversions and chops into several files if PDS format
MPRECOM	MPREC file TAGS(tag list) OUT(out file) Precomposer for PLI programs. Extracts marked lines from master source and outputs PLI source
MPREMES switches	MPREMES Concatenate continuation lines
MREMNUM	MREM trunc,check Removes line numbers from PLI source files (<trunc> char.) (<check> char. are checked to be digits)
MREPLACE	input[,output]/PRINT/FORMAT/UPPER Replaces old string by new string. Adjusts Documentation headers by option.
MSECTION	basename [/version] Check if a data base is mounted.
MSHOSYM	from DCL Procedure CSHOWSYM.COM Display the current value of a global symbol (wildcard)
MSHOW	GOOEXE:MSHOW <command> Show commands for use in spawned processes

MSTATUS	SHOW GOOSY STATUS proc1 [proc2 [proc3]] GSTATUS proc1 [proc2 [proc3]] Activate GOOSY status program. Equivalent to SHOW GOOSY STATUS command.
MSYMHLP	MSYM Reads a command procedure and generates help file.
MTMR	RUN GOO$EXE:MTMR DCL process with a subset of the transport manager functions.
MTRIM	switches Remove trailing spaces from source INPUT and writes result to destination OUTPUT
MTXTSORT	MTXTS*ORT input file,[output file] Sorts 2+ blocks in text file alphabetically.
MUAMODI	MUA <action string> <match string> Runs AUTHORIZE and modifies all user accounts (Called in CUAMODI.COM)
MUDIRO	GOOEXE:MUDIRO.EXE sorts a given directory list
MUTIL	– Activate different Activities using the command dispatcher.
MVMECMD	– VME command executor.
MVOICEX	MVOICE Extracts documentation blocks generated by MGENHEAD and generates a text module for U$TALK.
MWV	– inquires a diary, 'Wiedervorlage'

Appendix E

GOOSY Macros

$ACCU

$ACCU(type,base,dir,name,incr,dim,x1,x2,...)

PURPOSE	Accumulate spectrum
ARGUMENTS	
type	Type of spectrum (S,L,I or R)
base	Name of data base (plain text)
dir	Directory (plain text)
name	Data element name (plain text)
incr	Increment (expression)
dim	Dimensionality (1 or 2)
x1,x2..	Expressions to calculate the bin number (as many as dim)

FUNCTION Generates code to accumulate spectra. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

```
The spectrum must be located by $LOC(SPEC,...) The macro
expansion may be controlled to
     expand inline code
     expand fast inline code
     expand subroutine call
The fast inline code does NOT check freeze bits
                          does NOT set execute bits
                          does NOT increment counters
The modes can selected by the COMPILE switches:
        /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.
```

EXAMPLE

```
see example routine GOO$TEST:X$ANAL.PPL
    @INCLUDE $MACRO($MACRO);
    $ACCU(L,db,$spectrum,s1,1,1,x);
    $ACCU(R,db,$spectrum,s2,1,2,x,y);
    $ACCU(I,db,$spectrum,s3,1,2,x,y);
    $ACCU(S,db,$spectrum,s3,1,2,x,y);
```

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($ACCU)

$ACCU1

$ACCU1(type,base,dir,name,ind,incr,dim,x1,x2,...)

PURPOSE	Accumulate 1-dim. indexed spectrum
ARGUMENTS	
type	Type of spectrum (S,L,I or R)
base	Name of data base (plain text)
dir	Directory (plain text)
name	Data element name (plain text)
ind	Name index (expression)
incr	Increment (expression)
dim	Dimensionality (1 or 2)
x1,x2..	Expression to calculate bin number (as many as dim)
FUNCTION	Generates code to accumulate spectra. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

The spectrum must be located by $LOC1(SPEC,...) The macro expansion may be controlled to

```
     expand inline code
     expand fast inline code
     expand subroutine call
The fast inline code does NOT check freeze bits
                              does NOT set execute bits
                              does NOT increment counters
The modes are selected by the COMPILE switches:
        /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.
```

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL @INCLUDE $MACRO($MACRO); $ACCU1(L,db,$spectrum,tof,5,inc,1,x); $ACCU1(R,db,$spectrum,ede,7,inc,2,e,de); $ACCU1(I,db,$spectrum,ede,7,inc,2,e,de);
Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($ACCU1)

$ACCU2

$ACCU2(type,base,dir,name,i1,i2,incr,dim,x1,x2,...)

PURPOSE	Accumulate 2-dim. indexed spectrum
ARGUMENTS	
type	Type of spectrum (S,L,I or R)
base	Name of data base (plain text)
dir	Directory (plain text)
name	Data element name (plain text)
i1,i2	Name indices (expressions)
incr	Increment (expression)
dim	Dimensionality (1 or 2)
x1,x2..	Expression to calculuate bin number (as many as dim)
FUNCTION	Generates code to accumulate spectra Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

The spectrum must be located by $LOC2(SPEC,...) The macro expansion may be controlled to

expand inline code
expand fast inline code
expand subroutine call

The fast inline code does NOT check freeze bits
does NOT set execute bits
does NOT increment counters

The modes are selected by the COMPILE switches:
/FAST /CALL /MACRO (=default)
You must include $MACRO in the program.

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL

```
    @INCLUDE $MACRO($MACRO);
    $ACCU2(L,db,$spectrum,tof,2,1,1,1,t);
    $ACCU2(R,db,$spectrum,ede,6,10,inc,2,e,de);
    $ACCU2(I,db,$spectrum,ede,6,10,inc,2,e,de);
```

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($ACCU2)

$ATTACH

$ATTACH(type,base,access)

PURPOSE	Attach data base items
ARGUMENTS	
type	Type of item: BASE
base	Name of data base (plain text)
access	Access mode: W for write R for readonly
FUNCTION	This macro can be called during the initialization of programs accessing data bases. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.
REMARKS	You must include $MACRO in the program.

EXAMPLE	@INCLUDE $MACRO($MACRO);
	$ATTACH(BASE,db,W);
	IF ˆ STS$success THEN @RET(STS$value);

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($ATTACH)

$COND

$COND(type,base,dir,name,result,dim,x1,x2,...)

PURPOSE	Executes condition and returnes result.
ARGUMENTS	
type	Type of condition: MW MWI WC ANY INCL IDENT EXCL POLY
base	Name of data base (plain text)
dir	Directory of condition (plain text)
name	name of condition (plain text)
result	Result variable (BIT1 except BF31 for MW)
dim	Dimensionality (must be 1 for MW)
x1,x2...	Expression to be tested (as many as dim)
FUNCTION	Generates code to check condition. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

```
The condition must be located by $LOC(COND,...)
  You must include $MACRO in the program.
  Condition can be executed several times.
The macro expansion may be controlled to
    expand inline code
    expand fast inline code
    expand subroutine call
The fast inline code does NOT check freeze bits
                         does NOT set execute bits
                         does NOT increment counters
The modes are selected by the COMPILE switches:
      /FAST /CALL /MACRO (=default)
```

EXAMPLE see example routine GOO$TEST:X$ANAL.PPL

```
@INCLUDE $MACRO($MACRO);
$COND(WC,db,$condition,win,B_res,1,X);
$COND(WC,db,$condition,win,B_res,2,X,Y);
$COND(ANY,db,$condition,pat,B_res,1,X);
$COND(MW,db,$condition,multi,L_res,1,X);
$COND(MWI,db,$condition,multi,L_res,1,X);
$COND(POLY,db,$condition,pl,B_res,1,X,Y);
```

Version	1.01
Author	H.G.Essel
Last Update	27-NOV-1986
Include name	GOOINC($COND)

MW

Multi window, dim=1.
result is BIN FIXED(31)
Object must be BIN FLOAT(24). Result is the number of the LAST matching subwindow. The dimension parameter is ignored. All bits of the subwindows are set if true. If the subwindows overlap, the index of the last matching is returned. The order of subwindows is the order of checking. All subwindows are checked to set the result bits.

MWI

Multi window, dim=1.
result is BIN FIXED(31)
Object must be BIN FLOAT(24). Result is the number of the FIRST matching subwindow. The dimension parameter is ignored. NO bits of the subwindows are set. If the subwindows overlap, the index of the first matching is returned. The order of subwindows is the order of checking.
This type should be used if the subwindows do not
overlap, because checking is terminated after the first true subwindow.
In /FAST mode the condition result (index) cannot
be used in a subsequent dynamic list.

WC

Window, dim=1...4.
result is BIT(1) ALIGNED

Objects must be BIN FLOAT(24). Result is TRUE, if
all objects are inside their subwindow limits

INCL

Pattern condition, dim=1...4.
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern) = pattern
all subchecks must be true

ANY

Pattern condition, dim=1...4.
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern) ˆ = 0
all subchecks must be true

IDENT

Pattern condition, dim=1...4
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
object = pattern
all subchecks must be true

EXCL

Pattern condition, dim=1...4
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern = object)
all subchecks must be true

POLY

Polygon condition, dim=2

result is BIT(1) ALIGNED
Objects must be BIN FLOAT(24). Result is TRUE, if
point is inside the polygon

$COND1

$COND1(type,base,dir,name,ind,result,dim,x1,x2,...)

PURPOSE	Executes 1-dim. indexed condition and returnes result
ARGUMENTS	
type	Type of condition: MW MWI WC ANY INCL IDENT EXCL POLY
base	Name of data base (plain text)
dir	Directory of condition (plain text)
name	name of condition (plain text)
ind	Name index (expression)
result	Result variable (BIT1 except BF31 for MW)
dim	Dimensionality (must be 1 for MW)
x1,x2...	Expression to be tested (as many as dim)
FUNCTION	Generates code to check condition. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

```
The condition must be located by $LOC1(COND,...)
  Condition can be executed several times.
The macro expansion may be controlled to
    expand inline code
    expand fast inline code
    expand subroutine call
The fast inline code does NOT check freeze bits
                          does NOT set execute bits
                          does NOT increment counters
The modes are selected by the COMPILE switches:
      /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.
```

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL @INCLUDE $MACRO($MACRO); $COND1(WC,db,$condition,win,3,B_res,1,X); $COND1(WC,db,$condition,win,2,B_res,2,X,Y); $COND1(ANY,db,$condition,pat,6,B_res,1,X); $COND1(MW,db,$condition,multi,I,L_res,1,X); $COND1(POLY,db,$condition,poly,10,B_res,1,X);
Version	1.01
Author	H.G.Essel
Last Update	23-AUG-1988
Include name	GOOINC($COND1)

MW

Multi window, dim=1.
result is BIN FIXED(31)
Object must be BIN FLOAT(24). Result is the number
of the LAST matching subwindow. The dimension parameter is ignored. All bits of the subwindows are set if true. If the subwindows overlap, the index of the last matching is returned. The order of subwindows is the order of checking. All subwindows are checked to set the result bits.

MWI

Multi window, dim=1.
result is BIN FIXED(31)
Object must be BIN FLOAT(24). Result is the number of the FIRST matching subwindow. The dimension parameter is ignored. NO bits of the subwindows are set. If the subwindows overlap, the index of the first matching is returned. The order of subwindows is the order of checking.
This type should be used if the subwindows do not
overlap, because checking is terminated after the first true subwindow.
In /FAST mode the condition result (index) cannot
be used in a subsequent dynamic list.

WC

Window, dim=1...4.
result is BIT(1) ALIGNED

Objects must be BIN FLOAT(24). Result is TRUE, if
all objects are inside their subwindow limits

INCL

Pattern condition, dim=1...4.
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern) = pattern
all subchecks must be true

ANY

Pattern condition, dim=1...4.
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern) ^ = 0
all subchecks must be true

IDENT

Pattern condition, dim=1...4
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
object = pattern
all subchecks must be true

EXCL

Pattern condition, dim=1...4
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern = object)
all subchecks must be true

POLY

Polygon condition, dim=2

result is BIT(1) ALIGNED
Objects must be BIN FLOAT(24). Result is TRUE, if
  point is inside the polygon

$COND2

$COND2(type,base,dir,name,i1,i2,result,dim,x1,x2,..)

PURPOSE	Executes 2-dim. indexed condition and returnes result
ARGUMENTS	
type	Type of condition: MW MWI WC ANY INCL IDENT EXCL POLY
base	Name of data base (plain text)
dir	Directory of condition (plain text)
name	name of condition (plain text)
i1,i2	Name indices (expression)
result	Result variable (BF31 except BF31 for MW)
dim	Dimensionality (must be 1 for MW)
x1,x2...	Expression to be tested (as many as dim)
FUNCTION	Generates code to check condition. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

The condition must be located by $LOC2(COND,...)
  Condition can be executed several times.
The macro expansion may be controlled to
    expand inline code
    expand fast inline code
    expand subroutine call
The fast inline code does NOT check freeze bits
                          does NOT set execute bits
                          does NOT increment counters
The modes are selected by the COMPILE switches:
      /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.

EXAMPLE see example routine GOO$TEST:X$ANAL.PPL

```
@INCLUDE $MACRO($MACRO);
$COND2(WC,db,$condition,win,1,2,B_res,1,X);
$COND2(WC,db,$condition,win,7,1,B_res,2,X,Y);
$COND2(ANY,db,$condition,pat,3,2,B_res,1,X);
$COND2(MW,db,$condition,multi,1,1,L_res,1,X);
$COND2(POLY,db,$condition,poly,1,1,b_res,1,X);
```

Version 1.01

Author H.G.Essel

Last Update 23-AUG-1988

Include name GOOINC($COND2)

MW

Multi window, dim=1.
result is BIN FIXED(31)
Object must be BIN FLOAT(24). Result is the number
of the LAST matching subwindow. The dimension parameter is ignored. All bits of the subwindows are set if true. If the subwindows overlap, the index of the last matching is returned. The order of subwindows is the order of checking. All subwindows are checked to set the result bits.

MWI

Multi window, dim=1.
result is BIN FIXED(31)
Object must be BIN FLOAT(24). Result is the number of the FIRST matching subwindow. The dimension parameter is ignored. NO bits of the subwindows are set. If the subwindows overlap, the index of the first matching is returned. The order of subwindows is the order of checking.
This type should be used if the subwindows do not
overlap, because checking is terminated after the first true subwindow.
In /FAST mode the condition result (index) cannot
be used in a subsequent dynamic list.

WC

Window, dim=1...4.
result is BIT(1) ALIGNED

Objects must be BIN FLOAT(24). Result is TRUE, if
all objects are inside their subwindow limits

INCL

Pattern condition, dim=1...4.
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern) = pattern
all subchecks must be true

ANY

Pattern condition, dim=1...4.
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern) ^ = 0
all subchecks must be true

IDENT

Pattern condition, dim=1...4
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
object = pattern
all subchecks must be true

EXCL

Pattern condition, dim=1...4
result is BIT(1) ALIGNED
Objects must be BIT(32) ALIGNED. They are inverted
using the invert patterns stored in the condition
(object & pattern = object)
all subchecks must be true

POLY

Polygon condition, dim=2

result is BIT(1) ALIGNED
Objects must be BIN FLOAT(24). Result is TRUE, if
point is inside the polygon

$DE

$DE(base,dir,name,member)

PURPOSE	Data elements reference
ARGUMENTS	
base	Name of data base (plain text)
dir	Directory (plain text)
name	Name (plain text)
member	Member specification of structure to be accessed.
FUNCTION	From the first three arguments a pointer name is built. This pointer points to the member expresssion The pointer is declared by the $LOC macro. Its name is P$_base_directory_name.
REMARKS	The data base and pool must be attached. The data element must be located by $LOC. You must include $MACRO in the program.
EXAMPLE	@INCLUDE $MACRO($MACRO); $DE(db,data,d1,i_s_array(2,3))=0; X=$DE(db,par,d2,l_sa_struc.x(I))+5.; $DE(db,eva,d3,event.pattern)='0'B;
Version	1.01
Author	H.G.Essel
Last Update	16-Nov-1987
Include name	GOOINC($DE)

$DE1

$DE1(base,dir,name,index,member)

PURPOSE	Data elements reference
ARGUMENTS	
base	Name of data base (plain text)
dir	Directory (plain text)
name	Name (plain text)
index	Index expression for name array
member	Member specification of structure to be accessed.
FUNCTION	From the first three arguments a pointer name is built. This pointer points to the member expresssion The pointer is declared by the $LOC1 macro. Its name is P1$_base_directory_name(i).
REMARKS	The data base and pool must be attached. The data element must be located by $LOC1. You must include $MACRO in the program.
EXAMPLE	@INCLUDE $MACRO($MACRO); $DE1(db,data,d1,5,i_s_array(2,3))=0; X=$DE1(db,par,d2,2,l_sa_struc.x(I))+5.; $DE1(db,eva,d3,4,event.pattern)='0'B;
Version	1.01
Author	H.G.Essel
Last Update	16-Nov-1987
Include name	GOOINC($DE1)

$DE2

$DE2(base,dir,name,i1,i2,member)

PURPOSE	Data elements reference (2-dim)
ARGUMENTS	
base	Name of data base (plain text)
dir	Directory (plain text)
name	Name (plain text)
i1,i2	Two index expressions for name array
member	Member specification of structure to be accessed.
FUNCTION	From the first three arguments a pointer name is built. This pointer points to the member expresssion The pointer is declared by the $LOC2 macro. Its name is P2$_base_directory_name(i,k).
REMARKS	The data base and pool must be attached. The data element must be located by $LOC2. You must include $MACRO in the program.
EXAMPLE	@INCLUDE $MACRO($MACRO); $DE2(db,data,d1,5,4,i_s_array(2,3))=0; X=$DE2(db,par,d2,K,J,l_sa_struc.x(I))+5.; $DE2(db,eva,d3,K+4,I*(J-1),event.pattern)='0'B;
Version	1.01
Author	H.G.Essel
Last Update	16-Nov-1987
Include name	GOOINC($DE2)

$DETACH

$DETACH(type,base)

PURPOSE	Detach data base items
ARGUMENTS	
type	Type of item: BASE
base	Name of data base (plain text)
FUNCTION	This macro can be called at the end of programs accessing data bases. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.
REMARKS	You must include $MACRO in the program.

EXAMPLE	@INCLUDE $MACRO($MACRO);
	$DETACH(BASE,db);
	IF ^ STS$success THEN @RET(STS$value);

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($DETACH)

$LOC

$LOC(type,base,dir,name,access,descr)

PURPOSE Locate data elements for analysis

ARGUMENTS

type Type of data element:
SPEC spectrum
COND condition
DE general data element

base Name of data base (plain text)

dir Directory (plain text)

name Name (plain text)

access Access mode:
W for write
R for readonly

descr Data element type. will be checked. (plain text)

SPEC	L,I,R,S
COND	WC,PC,MW,POLY
DE	name of data element type (optional)

FUNCTION This routine must be called during the initialization of the analysis routine. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS The data base and pool must be attached.
You must include $MACRO in the program.

EXAMPLE see example routine GOO$TEST:X$ANAL.PPL

```
@INCLUDE $MACRO($MACRO);
$LOC(SPEC,db,$spectrum,tof,W,L);
```

```
$LOC(COND,db,$condition,w1,W,WC);
$LOC(DE,db,eva,event,W,SE$E1_1);
```

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($LOC)

SPEC

Any spectrum accessed by $ACCU must be located first by this macro:
    $LOC(SPEC,base,dir,name,W,t);
    $SECDEF must be included.
  Four pointers are declared for each spectrum:
   P$_base_directory_spectrum_t used by $ACCU
   P$_base_directory_spectrum_$H points to SE$SPHE
   P$_base_directory_spectrum_$A points to SE$SPDTT
   P$_base_directory_spectrum_$D points to SE$SPDti
    where t=I,L,S or R and i=1 or 2

COND

Any condition accessed by $COND must be located first by this macro:
    $LOC(COND,base,dir,name,W,t);
    $SECDEF must be included.
  Three pointers are declared for each condition:
   P$_base_directory_condition_t used by $COND
   P$_base_directory_condition_$H points to SE$COHE
   P$_base_directory_condition_$D points to SE$COxxx
  where xxx is a key for different condition types.
  and t=WC,PC,MW,POLY
   Command LIBLIS GOOTYP(SE$CO*) lists these names.

DE

Any data element to be accessed must be located first by this macro:
    $LOC(DE,base,dir,name,W,type);
    $SECDEF must be included.
  After that, the pointer to the data element is:
    P$_base_dir_name.

This pointer is declared as STATIC.
The length of the data element is returned in:
    L$_base_dir_name.
This Longword is declared STATIC.

$LOC1

$LOC1(type,base,dir,name,ll,ul,access,descr)

PURPOSE Locate 1-dim. data element arrays for analysis

ARGUMENTS

type Type of data element:
SPEC spectrum
COND condition
DE general data element

base Name of data base (plain text)

dir Directory (plain text)

name Name (plain text)

ll Boundary: lower limit (number)

ul Boundary: upper limit (number)

access Access mode:
W for write
R for readonly

descr Data element type. will be checked. (plain text)

SPEC	L,I,R,S
COND	WC,PC,MW,POLY
DE	name of data element type (optional)

FUNCTION This routine must be called during the initialization of the analysis routine. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS The data base and pool must be attached.
You must include $MACRO in the program.

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL @INCLUDE $MACRO($MACRO); $LOC1(SPEC,db,$spectrum,tof,1,5,W,L); $LOC1(COND,db,$condition,w1,1,10,W,WC); $LOC1(DE,db,eva,event,-2,5,W,SE$E1_1);
Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($LOC1)

SPEC

Any spectrum accessed by $ACCU must be located first by this macro:
 $LOC1(SPEC,base,dir,name,1,5,W,t);
 $SECDEF must be included.
Four pointers are declared for each spectrum:
 P1$_base_directory_spectrum_t(k) used by $ACCU1
 P1$_base_directory_spectrum_$H(k) to SE$SPHE
 P1$_base_directory_spectrum_$A(k) to SE$SPDTT
 P1$_base_directory_spectrum_$D(k) to SE$SPDti
 where t=I,L,S or R and i=1 or 2

COND

Any condition accessed by $COND must be located first by this macro:
 $LOC1(COND,base,dir,name,1,6,W,t);
 $SECDEF must be included.
Three pointers are declared for each condition:
 P1$_base_directory_condition_t(i) used by $COND1
 P1$_base_directory_condition_$H(i) to SE$COHE
 P1$_base_directory_condition_$D(i) to SE$COxxx
where xxx is a key for different condition types.
and t=WC,PC,MW,POLY
 Command LIBLIS GOOTYP(SE$CO*) lists these names.

DE

Any data element to be accessed must be located first by this macro:
 $LOC1(DE,base,dir,name,2,4,W,descr);

$SECDEF must be included.
After that, the pointer to the i-th data element is:
P1$_base_dir_name(i).
This pointer is declared as STATIC.

$LOC2

**$LOC2(type,base,dir,name,l1,u1,l2,u2
,access,descr)**

PURPOSE Locate 2-dim. data element arrays for analysis

ARGUMENTS

type Type of data element:
SPEC spectrum
COND condition
DE general data element

base Name of data base (plain text)

dir Directory (plain text)

name Name (plain text)

li,ui lower and upper boundaries of i-th dimension, i=1,2. Numbers are required here, no variables are allowed.

access Access mode:
W for write
R for readonly

descr Data element type. will be checked. (plain text)

SPEC	L,I,R,S
COND	WC,PC,MW,POLY
DE	name of data element type (optional)

FUNCTION This routine must be called during the initialization of the analysis routine. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS The data base and pool must be attached.
You must include $MACRO in the program.

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL @INCLUDE $MACRO($MACRO); $LOC2(SPEC,db,$spectrum,tof,1,5,2,4,W,L); $LOC2(COND,db,$condition,w1,1,10,1,5,W,WC); $LOC2(DE,db,eva,event,1,2,1,2,W,SE$E1_1);
Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($LOC2)

SPEC

Any spectrum accessed by $ACCU must be located first by this macro:
$LOC2(SPEC,base,dir,name,1,5,1,3,W,t);
$SECDEF must be included.
Four pointers are declared for each spectrum:
P2$_base_directory_spectrum_t(l,k) used by $ACCU
P2$_base_directory_spectrum_$H(l,k) to SE$SPHE
P2$_base_directory_spectrum_$A(l,k) to SE$SPDTT
P2$_base_directory_spectrum_$D(l,k) to SE$SPDti
where t=I,L,S or R and i=1 or 2

COND

Any condition accessed by $COND must be located first by this macro:
$LOC2(COND,base,dir,name,2,4,1,6,W,t);
$SECDEF must be included.
Three pointers are declared for each condition:
P2$_base_directory_condition_t(i,k) used by $COND2
P2$_base_directory_condition_$H(i,k) to SE$COHE
P2$_base_directory_condition_$D(i,k) to SE$COxxx
where xxx is a key for different condition types.
and t=WC,PC,MW,POLY
Command LIBLIS GOOTYP(SE$CO*) lists these names.

DE

Any data element to be accessed must be located first by this macro:
$LOC2(DE,base,dir,name,-3,5,2,5,W,descr);

$SECDEF must be included.
The pointer to the i,k-th data element is:
P2$_base_dir_name(i,k).
This pointer is declared as STATIC.

$MACRO

```
@INCLUDE $MACRO($MACRO)
```

PURPOSE Initialize analysis macros

FUNCTION Must be included if any analysis macro is called like $LOCx, $CONDx or $ACCUx.

$SPEC

$SPEC(type,base,dir,name,value,dim,x1,x2,...)

PURPOSE	Set spectrum channel
ARGUMENTS	
type	Type of spectrum (L,I or R)
base	Name of data base (plain text)
dir	Directory (plain text)
name	Data element name (plain text)
value	Channel value (expression)
dim	Dimensionality (1 or 2)
x1,x2..	Expressions to calculate the bin number (as many as dim)
FUNCTION	Generates code to set spectrum channel. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

```
The spectrum must be located by $LOC(SPEC,...) The macro
expansion may be controlled to
     expand inline code
     expand fast inline code
     expand subroutine call
The fast inline code does NOT check freeze bits
                              does NOT set execute bits
                              does NOT increment counters
The modes can selected by the COMPILE switches:
        /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.
```

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL

```
@INCLUDE $MACRO($MACRO);
$SPEC(L,db,$spectrum,s1,1,1,x);
$SPEC(R,db,$spectrum,s2,1,2,x,y);
$SPEC(I,db,$spectrum,s3,1,2,x,y);
```

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($SPEC)

$SPEC1

$SPEC1(type,base,dir,name,ind,value,dim,x1,x2,...)

PURPOSE	Set channel in 1-dim. indexed spectrum
ARGUMENTS	
type	Type of spectrum (L,I or R)
base	Name of data base (plain text)
dir	Directory (plain text)
name	Data element name (plain text)
ind	Name index (expression)
value	Channel value (expression)
dim	Dimensionality (1 or 2)
x1,x2..	Expression to calculate bin number (as many as dim)
FUNCTION	Generates code to set spectrum channel. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

```
The spectrum must be located by $LOC1(SPEC,...) The macro
expansion may be controlled to
    expand inline code
    expand fast inline code
    expand subroutine call
The fast inline code does NOT check freeze bits
                          does NOT set execute bits
                          does NOT increment counters
The modes are selected by the COMPILE switches:
      /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.
```

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL

```
@INCLUDE $MACRO($MACRO);
$SPEC1(L,db,$spectrum,tof,5,inc,1,x);
$SPEC1(R,db,$spectrum,ede,7,inc,2,e,de);
$SPEC1(I,db,$spectrum,ede,7,inc,2,e,de);
```

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($SPEC1)

$SPEC2

$SPEC2(type,base,dir,name,i1,i2,value,dim,x1,x2,...)

PURPOSE	Set channel in 2-dim. indexed spectrum
ARGUMENTS	
type	Type of spectrum (L,I or R)
base	Name of data base (plain text)
dir	Directory (plain text)
name	Data element name (plain text)
i1,i2	Name indices (expressions)
value	Channel value (expression)
dim	Dimensionality (1 or 2)
x1,x2..	Expression to calculuate bin number (as many as dim)
FUNCTION	Generates code to set spectrum channel. Plain text means that these arguments must not be enclosed in quotes and must not be PL/1 variables.

REMARKS

```
The spectrum must be located by $LOC2(SPEC,...) The macro
expansion may be controlled to
     expand inline code
     expand fast inline code
     expand subroutine call
The fast inline code does NOT check freeze bits
                              does NOT set execute bits
                              does NOT increment counters
The modes are selected by the COMPILE switches:
        /FAST /CALL /MACRO (=default)
  You must include $MACRO in the program.
```

EXAMPLE	see example routine GOO$TEST:X$ANAL.PPL

```
@INCLUDE $MACRO($MACRO);
$SPEC2(L,db,$spectrum,tof,2,1,1,1,t);
$SPEC2(R,db,$spectrum,ede,6,10,inc,2,e,de);
$SPEC2(I,db,$spectrum,ede,6,10,inc,2,e,de);
```

Version	1.01
Author	H.G.Essel
Last Update	27-AUG-1985
Include name	GOOINC($SPEC2)

ADD_MSG

@ADD_MSG(errorcode,arg1,arg2,arg3)

PURPOSE	accomplish the error message belonging to the errorcode by the specified arguments and write the message on the internal error- message stack.
ARGUMENTS	
errorcode	name of error
argi	parameters for the message text
Include name	-

Description

CALLING	@ADD_MSG(errorcode,arg1,arg2,arg3)
ARGUMENTS	
errorcode	name of error for which the message should be written
argi	parameters for the message text. Subsequent !AS in the message text as it is defined , will be replaced by specified arguments. Arguments can be omitted from the right.
FUNCTION	The message belonging to the specified error code will be retrieved and accomplished by the given additional arguments if necessary. THe complete message will then be written to the internal message stack, where it is held for further processing (see @DMP_CLR_MSG, @PUT_CLR_MSG).
REMARKS	the arguments argi are of type CHAR VAR, however on the VAX they can be of any type.
EXAMPLE	@ADD_MSG(XUTIL_NOOUTPUT,'U_OUT','CV_LONG'); will generate the following message and write it to the message stack: 'U_OUT tried to output CV_LONG, but did not find a valid control pattern'

BYTE

@BYTE(integer)

PURPOSE	returns the ASCII(EBCDIC) character whose code is equivalent to the given integer.
ARGUMENTS	
integer	integer number
Include name	-

Description

CALLING	@BYTE(integer)
ARGUMENTS	
integer	integer number, may be BIN FIXED(15) or BIN FIXED(31
FUNCTION	the binary representation of the argument <integer> is interpreted as character in ASCII (VAX) or EBCDIC (IBM) format.
REMARKS	If the given number exceeds the valid range of 0 to 255, an error will be signaled.
EXAMPLE	C=@BYTE(32); a blank will be returned on the VAX, C=@BYTE(64); a blank will be returned on the IBM, a '@' on the VAX.

CALL

@CALL procedure

PURPOSE	performs a function call
ARGUMENTS	
procedure	procedure name with arguments
Include name	-

Description

CALLING	@CALL procedure
ARGUMENTS	
procedure	name of procedure followed by the arguments in brackets
FUNCTION	'@CALL' will be replaced by the string 'STS$VALUE=' Care will be taken of the existance of the declaration for the errors (an include of $STSDEF)
REMARKS	implemented as a string variable.
EXAMPLE	@CALL U$PRTCL(CV,U$MPRTTERM);

CLR_MSG

@CLR_MSG

PURPOSE	clear the internal message stack on
ARGUMENTS	
Include name	-

Description

CALLING	@CLR_MSG
ARGUMENTS	
FUNCTION	The internal message stack will be cleared.
REMARKS	-

EXAMPLE

```
@CALL MYSUB(CV_NAME,I_NUMBER);
  IF STS$SUCCESS THEN @CLR_MSG;
```

DCL_MSG

@DCL_MSG(errorname);

PURPOSE	declaration of error
ARGUMENTS	
error name	name of error to be declared
Include name	-

Description

CALLING	@DCL_MSG(<error name>);
ARGUMENTS	
error name	name of error, looks like X<fac>_<name> where <fac> is a facility key word and <name> is the name of the specific error.
FUNCTION	the error is declared as GLOBAL REF VALUE
REMARKS	do not declare several errors in one declaration @DCL_MSG
EXAMPLE	@DCL_MSG(XTEST_ER);

DMP_CLR_MSG

```
@DMP_CLR_MSG
```

PURPOSE write the internal message stack on the screen

ARGUMENTS

Include name -

Description

CALLING @DMP_CLR_MSG

ARGUMENTS

FUNCTION The internal message stack will be written to the terminal. The stack will be cleared. All messages will be written, regardless of the message profile set (good for test purposes, for message profile dependent output see @PUT_CLR_MSG).

REMARKS -

EXAMPLE
```
@CALL MYSUB(CV_NAME,I_NUMBER);
  IF ^ STS$SUCCESS THEN @DMP_CLR_MSG;
```

ENTRY

label: @ENTRY

PURPOSE	remember name of entry
ARGUMENTS	
label	name of the entry
Include name	-

Description

CALLING	label: @ENTRY
ARGUMENTS	
label	name of the entry
FUNCTION	The name of the entry (label) will be memorized until a redefinition (via @ENTRY or @PROCEDURE) takes place. This name will be used as a prefix for all error messages.
REMARKS	The syntax will be changed into @ENTRYPLI(label) in the near future. However the old form will be understood.
EXAMPLE	S$EXAE:@ENTRY(I_1) RETURNS(BIN FIXED(31)) ;

INCLUDE

@INCLUDE lib(member)

PURPOSE	include PPL code
ARGUMENTS	
lib	library
member	module in library (member of PDS)
Include name	-

Description

CALLING	@INCLUDE lib(member)
ARGUMENTS	
library	DEC library from which a module should be included (not yet implemented), or DD-name which is related by an ALLOC-statement (IBM-MVS). If omitted, <member> is interpreted as file specification (VAX only).
member	module in library(n.y.i.) or member of PDS or VAX file specification.
FUNCTION	The specified source code is included.
REMARKS	The VAX library handling is not yet implemented.
EXAMPLE	@INCLUDE $MACRO(U$PRTCL); the declaration of the routine U$PRTCL is included.

LOCAL_ERROR

@LOCAL_ERROR()

PURPOSE	resignals errors from lower procedure levels
ARGUMENTS	
Include name	-

Description

CALLING @LOCAL_ERROR()

ARGUMENTS

FUNCTION An On-unit, written to catch errors which happen in the local routine ,catches also errors from lower level procedures. @LOCAL_ERROR() will catch in that on-unit the error from lower routines, and will resignal them to higher levels.

REMARKS -

EXAMPLE

```
ON FIXEDOVERFLOW BEGIN;
   @LOCAL_ERROR();
   @CALL U$PRTCL('fixed overflow in my routine',
                U$M_PRTT);
END /* of ON FIXEDOVERFLOW */;
```

ON_ANY_E

@ON_ANY_E(u_cleanup)

PURPOSE	catches all signaled errors, calls <u_cleanup> before resignaling the error
ARGUMENTS	
u_cleanup	routine to be called after detecting the error and before resignaling
Include name	-

Description

CALLING @ON_ANY_E(u_cleanup)

ARGUMENTS

u_cleanup Routine which will be called when handling the signaled error. Arguments may be passed .

FUNCTION All signaled error will be detected by @ON_ANY_E. Following actions will take place:

If specified the routine u_cleanup will be called.
This routine serves to make things undone which has previously been performed in the current routine, e.g. free allocated space, close opened files a.s.o.

Any error during the clean-up will be caught by an
internal On-unit and will be resignaled as an unrecoverable fatal error which passes all higher on-units.

The message related to the occurred error will be
written on the internal error stack. The text will contain the name of current routine if the macros @PROCEDURE or @ENTRY are used.

Depending on the severity, the error will be
resignaled or converted to a reported error (RETURN(errorcode).

Here: error of severity E (error) and less will
not be resignaled, but a non local GOTO will be performed and the current routine will return the error code to the calling routine.

REMARKS Due to the lack of several severities in the system commands on the IBM, conversions to reported errors will not take place on these computers. All errors will then be resignaled.

EXAMPLE @ON_ANY_E(U_CLUP(I_COUNT));
the above described actions will take place. An
internal subroutine U_CLUP is called from within the standard On-unit, the argument I_COUNT is passed.

ON_ANY_F

@ON_ANY_F(u_cleanup)

PURPOSE	catches all signaled errors, calls <u_cleanup> before resignaling the error
ARGUMENTS	
u_cleanup	routine to be called after detecting the error and before resignaling
Include name	-

Description

CALLING @ON_ANY_F(u_cleanup)

ARGUMENTS

u_cleanup Routine which will be called when handling the signaled error. Arguments may be passed .

FUNCTION All signaled error will be detected by @ON_ANY_F.
If specified the routine u_cleanup will be called.
This routine serves to make things undone which has previously been performed in the current routine, e.g. free allocated space, close opened files ,...
Any error during the clean-up will be caught by an internal On-unit and will be resignaled as an unrecoverable fatal error
The message related to the occurred error will be written on the internal error stack. The text will contain the name of current routine if the macros @PROCEDURE or @ENTRY are used.
Depending on the severity, the error will be resignaled or converted to a reported error (RETURN(errorcode).
Here: all errors will not be resignaled, but a non local GOTO will be performed and the current routine will return the error code to the calling routine.

REMARKS Due to the lack of several severities in the system commands on the IBM, conversions to reported errors will not take place on these computers. All errors will then be resignaled.

EXAMPLE @ON_ANY_F(U_CLUP(I_COUNT));
the above described actions will take place. An
internal subroutine U_CLUP is called from within the standard On-unit, the argument I_COUNT is passed.

ON_ANY_W

@ON_ANY_W(u_cleanup)

PURPOSE	catches all signaled errors, calls <u_cleanup> before resignaling the error
ARGUMENTS	
u_cleanup	routine to be called after detecting the error and before resignaling
Include name	-

Description

CALLING @ON_ANY_W(u_cleanup)

ARGUMENTS

u_cleanup Routine which will be called when handling the signaled error. Arguments may be passed .

FUNCTION All signaled error will be detected by @ON_ANY_W. Following actions will take place:
If specified the routine u_cleanup will be called.
This routine serves to make things undone which has previously been performed in the current routine, e.g. free allocated space, close opened files ,...
Any error during the clean-up will be caught by an
internal On-unit and will be resignaled as an unrecoverable fatal error which passes all higher on-units.
The message related to the occurred error will be
written on the internal error stack. The text will contain the name of current routine if the macros @PROCEDURE or @ENTRY are used.
Depending on the severity, the error will be
resignaled or converted to a reported error (RETURN(errorcode).
Here: error of severity W (warning) and less will
not be resignaled, but a non local GOTO will be performed and the current routine will return the error code to the calling routine

REMARKS Due to the lack of several severities in the system commands on the IBM, conversions to reported errors will not take place on these computers. All errors will then be resignaled.

EXAMPLE @ON_ANY_W(U_CLUP(I_COUNT));
the above described actions will take place. An
internal subroutine U_CLUP is called from within the standard On-unit, the argument I_COUNT is passed.

PROCEDURE

<label>:@PROCEDURE

PURPOSE	remembers the name of the current module
ARGUMENTS	
Include name	-

Description

CALLING	<label>:@PROCEDURE
ARGUMENTS	
FUNCTION	The name of the current procedure is taken from the name of <label> and will later be inserted in all error messages uttered in this module.
REMARKS	The syntax of this macro will later be changed into @PROCPLI(<label>)
EXAMPLE	U$PRTCL: @PROCEDURE (CV_OUT,B32) RETURNS(BIN FIXED(31));

PUT_CLR_MSG

@PUT_CLR_MSG

PURPOSE	write the internal mesage stack on the screen
ARGUMENTS	
Include name	-

Description

CALLING	@PUT_CLR_MSG
ARGUMENTS	
FUNCTION	The internal message stack will be written to the The message profile, as set by a call to S$MSPRO will be considered.
REMARKS	-
EXAMPLE	@CALL MYSUB(CV_NAME,I_NUMBER); IF ^ STS$SUCCESS THEN @PUT_CLR_MSG;

RANK

@RANK(char)

PURPOSE returns a BIN FIXED (15) number which corresponds to the input character <char>.

ARGUMENTS

char character whose binary representation will be inter- preted

Include name -

Description

CALLING @RANK(char)

ARGUMENTS

char single character, input

FUNCTION the binary representation of the input character will be interpreted as an integer number, put into the low order byte of a BIN FIXED(15) number, which will then be returned.

REMARKS has the same functionality as the VAX builtin function RANK.

EXAMPLE

```
IF @RANK(SUBSTR(CV_NAME1,1,1))>
     @RANK(SUBSTR(CV_NAME2,1,1) THEN DO;
```

REPEAT

@REPEAT(cv,i_repeat)

PURPOSE	return the string cv concatenated to cv i_repeat times
ARGUMENTS	
cv	character string
i_repeat	number of concatenations

Description

FUNCTION	The string <cv> will be concatenated to itself I_repeat times, the result will be returned. Note: the resulting string contains one time CV more than the result of the VAX-PLI builtin function COPY.
Example	CV=@REPEAT('ei',3) ; CV gets the value 'eieieiei'.
File name	GOOMINC(REPEAT)
Dataset	-
Version	1.01
Author	K.Winkelmann
Last Update	24-JUN-1985

RET

@RET(errorcode)

PURPOSE	returns the error code to the calling procedure,
ARGUMENTS	
errorcode	name of error or number
Include name	-

Description

CALLING	@RET(errorcode)
ARGUMENTS	
errorcode	name of error
FUNCTION	@RET(errorcode) will be substituted by RETURN(errorcode)
REMARKS	-
EXAMPLE	@RET(1) successful completion

RET_ADD_MSG

@RET_ADD_MSG(errorcode,arg1,arg2,arg3)

PURPOSE	return and write specified message onto error stack
ARGUMENTS	
errorcode	name of error
argi	parameters for the related message
Include name	-

Description

CALLING	@RET_ADD_MSG(errorcode,arg1,arg2,arg3)
ARGUMENTS	
errorcode	name of error or number to be returned to calling procedure
argi	arguments which will be substituted in the message text
FUNCTION	the message text related to the given error code will be retrieved, parameters will be substituted and the accomplished text is written to the internal error stack. Then the procedure returns the error number to the calling procedure.
REMARKS	The syntax will be changed later like @RET_ADD_MSG errorcode arg1 arg2 arg3
EXAMPLE	IF ^ STS$SUCCESS THEN @RET_ADD_MSG(STS$VALUE);

RET_SET_MSG

@RET_SET_MSG(errorcode,arg1,arg2,arg3)

PURPOSE	return and write specified message onto error stack
ARGUMENTS	
errorcode	name of error
argi	parameters for the related message
Include name	-

Description

CALLING	@RET_SET_MSG(errorcode,arg1,arg2,arg3)
ARGUMENTS	
errorcode	name of error or number to be returned to calling procedure
argi	arguments which will be substituted in the message text
FUNCTION	the message text related to the given error code will be retrieved, parameters will be substituted and the accomplished text is written to the internal error stack after it has been cleared. Then the procedure returns with the error number to the calling procedure. This macro is useful if a new error message makes previous ones obsolete.
REMARKS	The syntax will be changed later like @RET_SET_MSG errorcode arg1 arg2 arg3
EXAMPLE	IF ^ STS$SUCCESS THEN @RET_SET_MSG(STS$VALUE);

SIZE

```
@SIZE(reference)
```

PURPOSE	returns number of bytes allocated to the referenced variable
ARGUMENTS	
reference	name of variable whose size is wanted

Description

FUNCTION	The number of bytes allocated for the referenced variable <reference> is returned (VAX-PLI builtin function SIZE).
File name	GOOMINC(SIZE)
Dataset	-
Version	-
Author	K.Winkelmann
Last Update	24-JUN-1985

STORAGE

@STORAGE(reference)

PURPOSE	returns number of bytes allocated to the referenced variable
ARGUMENTS	
reference	name of variable whose size of storage is wanted

Description

FUNCTION	The number of bytes allocated for the referenced variable <reference> is returned (IBM-PLI builtin function STORAGE).
File name	GOOMINC(STORAGE)
Dataset	-
Version	-
Author	K.Winkelmann
Last Update	24-JUN-1985

TRACE_MSG

@TRACE_MSG(errorcode,arg1,arg2,arg3)

PURPOSE	Write e trace message to the internal error stack. The errorcode is normally returned by another routine signaling an error.
ARGUMENTS	
errorcode	name of error
argi	parameters for the message text. Normally not used.
Include name	-

Description

CALLING	@TRACE_MSG(errorcode,arg1,arg2,arg3)
ARGUMENTS	
errorcode	name of error for which the trace message should be written
argi	parameters for the message text. Not used, if the errorcode was returned by a GOOSY routine call.
FUNCTION	If the message belonging to the specified error code is already on the error stack, a trace message is added.
REMARKS	the arguments argi are of type CHAR VAR, however on the VAX they can be of any type.

EXAMPLE

```
@CALL U$xxx();
  IF ^ STS$SUCCESS THEN DO;
      @TRACE_MSG(STS$VALUE);
      GOTO ERROR;
  END;
```
U$xxx returned an error. Write trace message and handle error.

TRIM

@TRIM(cv_string,cv_lead,cv_trail)

PURPOSE	remove leading and/or trailing characters from a string
ARGUMENTS	
cv_string	string to be trimmed
cv_lead	set of characters to be removed from left
cv_trail	set of characters to be removed from right
Include name	-

Description

CALLING @TRIM(cv_string,cv_lead,cv_trail)

ARGUMENTS

cv_string input string to be trimmed

cv_lead set of characters, each of them will be removed from the beginning

cv_trail set of characters, each of them will be removed from right. If cv_lead or cv_trail will be ommitted, they are assumed to be ' '(blank).

FUNCTION The TRIM function returns a character string that consists of the input string with specified characters removed from the left and right.TRIM takes either one or three arguments. If you supply second and third arguments, TRIM removes characters specified by those arguments from the left and right or the string, respectively.

REMARKS corresponds to the VAX builtin function TRIM.

EXAMPLE CV=@TRIM(' go to hell !!!!!!!!!!!',' ',' !');
after execution, CV will have the value
CV='go to hell' . CV=@TRIM(' the red rose ');
leads to CV='the red rose' .

Appendix F

GOOSY Data Formats

F.1 Introduction

F.1.1 Buffers

The GOOSY dump file format defines the structure of

1. data streams between the processors and processes controlled by GOOSY, e.g. the frontend equipment and the GOOSY processes,
2. dumps of data produced by GOOSY for later analysis or exchange of data between GOOSY and other systems.

The smallest entities of data, which are transported by GOOSY in the sense mentioned above, are called *buffers*. Presently these buffers have a fix length of either 16, 8 or 4 KByte. On disk, the buffers are stored in one RMS record, on tape several buffers can be stored into one tape record.

F.1.2 Buffer Files

If GOOSY buffers are dumped to files, the first buffer may be a file header buffer (see section F.4.2)! If the file is written to a tape, the tape is labled by **ANSI tape labels** as described in the ANSI standard (American National Standard X3.27-1978). In the appendix there is an overview of the implementations of this standard on DEC VAX/VMS and IBM MVS/XA. In general, GOOSY uses DEC's standard RMS file formats. The GOOSY files contain fixed length records.

F.1.3 Message Control Blocks

The GOOSY MCB format defines the structure of

1. control data streams between the processors and processes controlled by GOOSY, e.g. the frontend equipment and the GOOSY processes.

F.1.4 Glossary

byte means: 8–bit–sequence

word means: 2 bytes

longword means: 4 bytes.

buffer element Whole buffer or part of a buffer.

buffer element header unified structure keeping information about the trailing buffer element data.

buffer element data Data of any structure including other buffer elements. Always preceded by a buffer element header.

event Data describing one physical event. Events are buffer elements in standard buffers. There are, however, buffers containig events without headers (nonstandard buffers). Events may be composed of subevents.

If not otherwise stated:

All length fields are given in 16–bit word units. One box line in the structure figures represents a 32 bit word. Offsets are given in bytes. The order of bits, bytes, and words is always from the right to the left, i.e. from the least to the most significant bit or byte, as the VAX processes them. All character string fields are written with 7–bit ASCII coding.

Byte Order: Between machines with different byte ordering a longword swap must be performed. All Structures in this manual refer to the VAX byte ordering (little endian: least significant bit is in byte with lowest address). Big endian machines must use structure declarations with swapped words and bytes.

F.2 Message Control Block Structure

Control information between the VAX computers and the VME processors is packed in message control blocks. These are composed of a header and a message field. The message field contains a message header and a GOOSY buffer. The fields in the header are used on the local modules. No

Message control block

31 28 24 20 16	12 8 4 0	Offset
Queue forward pointer		Header 0
Queue back pointer		4
Link control block pointer		8
MCB protocol state	MCB queue state	12
transfer length	status code	(control block) 16
Device status		20
AST address		24
transfer length	status code	(control block) 28
Device status		32
AST address		36
transfer length	status code	(control block) 40
Device status		44
AST address		48
Pointer to auxiliary structure		52
Pointer to back slice		56
Pointer to forward slice		60
Pointer to acknowledge		64
Pointer to next slice		68
length of rest slice		72
Length of data in bytes		76
Length of message		80
Message subtype	Message type	mess.head. 84
Transaction number		88
Flags		92
Acknowledge status		96
Length of data in bytes		100
Begin of data		Data 104

Figure F.1: Message Control Block Structure

information is transferred. The message header contains information which is transferred. The structure is found in GOOINC(SN$MCB):

```
DCL 1 SN$MCB            BASED(P_SN$MCB),
2 SN$MCB_CTL,                           /* Control part */
  3 PN$MCB_NMCB(2)      POINTER,        /* Queue link */
  3 PN$MCB_LCB          POINTER,        /* LCB backpointer */
  3 IN$MCB_QSTATE       BIN FIXED(15),  /* MCB queue state */
  3 IN$MCB_PSTATE       BIN FIXED(15),  /* MCB protocol state */
  3 SN$MCB_PIOSB,                         /* IOSB used for NET-QIO's */
    4 IN$MCB_PIOSB_STAT BIN FIXED(15),  /* Operation status */
    4 IN$MCB_PIOSB_LGT  BIN FIXED(15),  /* Transfer length */
    4 LN$MCB_PIOSB_AUX  BIN FIXED(31),  /* Device specific information */
  3 EN$MCB_PAST         ENTRY(POINTER)  /* Completion AST */
                        RETURNS(BIN FIXED(31))
                        VARIABLE,
  3 SN$MCB_LIOSB,                         /* IOSB used for NET-QIO's */
    4 IN$MCB_LIOSB_STAT BIN FIXED(15),  /* Operation status */
    4 IN$MCB_LIOSB_LGT  BIN FIXED(15),  /* Transfer length */
    4 LN$MCB_LIOSB_AUX  BIN FIXED(31),  /* Device specific information */
  3 EN$MCB_LAST         ENTRY(POINTER)  /* Completion AST */
                        RETURNS(BIN FIXED(31))
                        VARIABLE,
  3 SN$MCB_TIOSB,                         /* IOSB used for NET-QIO's */
    4 IN$MCB_TIOSB_STAT BIN FIXED(15),  /* Operation status */
    4 IN$MCB_TIOSB_LGT  BIN FIXED(15),  /* transfer length */
    4 LN$MCB_TIOSB_AUX  BIN FIXED(31),  /* Device specific information */
  3 EN$MCB_TAST         ENTRY(POINTER)  /* Completion AST */
                        RETURNS(BIN FIXED(31))
                        VARIABLE,
  3 PN$MCB_APPL         POINTER,        /* Pointer to application DSC */
  3 PN$MCB_MCB_BACK     POINTER,        /* MCB backpointer for slicing */
  3 PN$MCB_MCB_FORW     POINTER,        /* MCB forward pointer for slicing */
  3 PN$MCB_MCB_ACKN     POINTER,        /* MCB pointer to acknowledge */
  3 PN$MCB_BUF_PTR      POINTER,        /* Point to next slice */
  3 LN$MCB_BUF_LGT      BIN FIXED(31),  /* Length of rest slice */
  3 LN$MCB_ALLOC_SIZE   BIN FIXED(31),  /* Allocation size */
  3 LN$MCB_MSG_SIZE     BIN FIXED(31),  /* Total message size */
                                        /* Header plus data part send */
2 SN$MCB_MSG,                           /* Total message */
  3 SN$MCB_HDR,                         /* Message header */
    4 IN$MCB_MSG_TYPE   BIN FIXED(15),  /* Message type */
    4 IN$MCB_MSG_SUBTYPE BIN FIXED(15), /* Message sub-type */
    4 LN$MCB_TSN        BIN FIXED(31),  /* Transaction number */
    4 BN$MCB_MODE       BIT(32) ALIGNED,/* Flags */
    4 LN$MCB_STAT_ACKN  BIN FIXED(31),  /* Acknowledge status */
    4 LN$MCB_DATA_SIZE  BIN FIXED(31),  /* Data size */
  3 SN$MCB_DATA,                        /* Message data */
    4 IN$MCB_DATA(1 $MCB_DATA REFER(LN$MCB_ALLOC_SIZE))
                        BIN FIXED(7);   /* Message data array */
```

F.3 Buffer Structure

F.3.1 Standard Buffers

- **Buffer Element**
  A GOOSY buffer contains an arbitrary number of buffer elements. Buffer elements, which are *not* known to GOOSY are invalid and rejected. Any buffer element is composed of two parts:

- **Buffer Element Header**
  Headers work like envelopes for data. Examples for headers are the buffer header (see section F.3.1) and the event header (section F.3.4). The header specifies the type and size of the following data.

- **Buffer Element Data**
  Arbitrary structured data. The structure may contain other buffer elements. The type specified in the buffer element header must always uniquely define the kind of data following.

Examples of buffer elements are the buffer itself, GOOSY events and GOOSY subevents. Others are time stamps, spectra etc.. Figure F.2 shows the buffer structure. One can see the nested structures. The headers always contain a type/subtype number combination and the word length of the following data. The type/subtype numbers are unique for a certain data structure. All modules processing buffers can check if a buffer element has the correct type. If not, it may just skip the element, output messages or skip the buffer.

F.3.2 Nonstandard Buffers

Structures, which are defined by external processors or by the hardware of a frontend system are called *external structures*. In standard buffers external structures are always enveloped by headers. These headers must be added by the frontend processors. An example is the event type 1 as described in section F.5.3, a structure, which is created by the SILENA 4418x ADC–System. If external structures without header are copied directly into a buffer, this buffer has no standard format. Examples of such external structures are the SILENA (section F.7.1) and FERA (section F.7.2) event structures, if they are not preprocessed by a frontend processor adding a header.

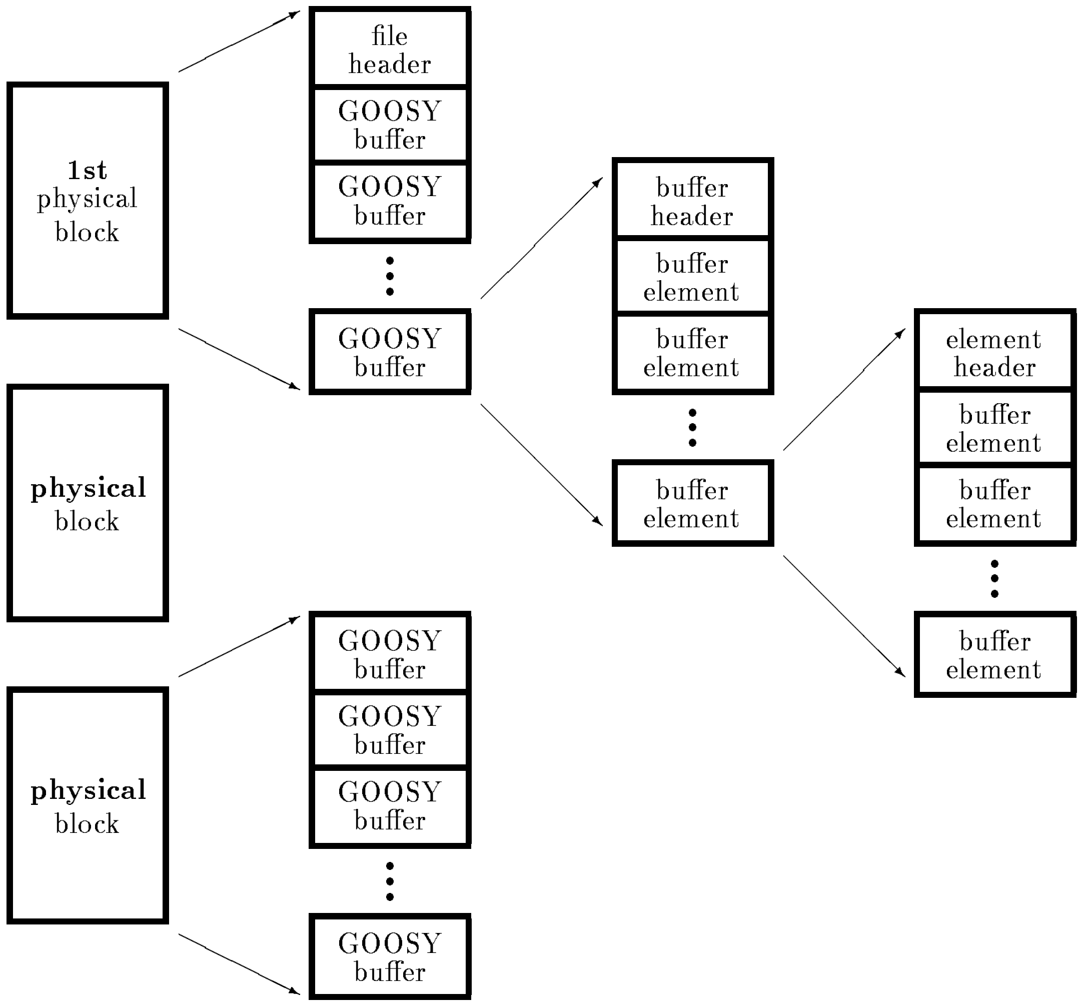


Figure F.2: The GOOSY data structures of a listmode dump file.

F.3.3 Buffer Header

Buffer Header

31 28 24	20 16	12 8 4 0	Offset
Length of data field (buffer without header)			0
Sub Type		Type	4
Fragment begin	Fragment end	Used Length of Data field	8
Buffer Number for this Type - Sub Type			12
Number of Buffer Elements			16
Current Index			20
Time stamp VMS 64 bit format			24
Time stamp VMS 64 bit format			28
Byte order tag			32
Length of last event			32
2 Longwords reserved			32
2nd Data word		1st Data word	48
. . .			

Figure F.3: Buffer Header Structure

The total length of the buffer header is 48 bytes.

Length of data field Length of the buffer without this buffer header in 16–bit words.

(BIN FIXED (31)).

Type A number specifying the buffer type.

(BIN FIXED (15)).

Sub Type A number specifying the buffer subtype.

(BIN FIXED (15)).

Used Length of Data Field Number of 16–bit words actually used in the Data field in this buffer.

(BIN FIXED (15)).

Fragment begin If this byte is= 1, the buffer contains a fragment (the first part of a buffer element, which is not complete) at the end of the buffer. The fragment is missing its trailing part, which has to be found in the following buffer of the same type and subtype.

(BIT(8)).

Fragment end If this byte is= 1, the buffer contains a fragment (the rest of a buffer element which is not complete) at the begin of the buffer. The fragment is missing its first part, which had to be found in the preceding buffer of the same type and subtype.

(BIT(8)).

Number of buffer elements This number is needed to decide in the case of fragment begin and fragment end, if there are two different fragments or only one fragment. A fragment is counted like a buffer element.

(BIN FIXED (31)).

Buffer Number A current number of buffers of the same type.

(BIN FIXED (31)).

Current Index A longword to store the index of the last processed event. This filed can be used by routines processing the buffer to store the index of the last processed buffer element. If the buffer is stored on disk or tape this field must be zero or 1.

(BIN FIXED (31)).

Time stamp A quadword for the system time in VAX/VMS binary format. This is the number of 100-nanoseconds since 17–Nov–1858 00:00.

(BIT(64)).

Byte order tag The creator of the buffer writes a 1 here. Each program processing the buffer must check this field. If it founds a 1, byte ordering is OK, if not, a longword swap must be performed.

(BIN FIXED (31)).

Length of last event When the last event in the buffer is a fragment, the length field in the event header keeps the size of the fragment. The length of the total event is kept in the buffer header.

(BIN FIXED (31)).

2 Free Longwords Reserved

((2) BIN FIXED(31)).

Data Words The Data Field of the buffer has a length specified by "Length of Data field", where only those words are used for data as specified in "Used Length of Data Field". The structure of the "Data Words" field is specified by buffer type and subtype.

(any).

Structure Declaration

The PL/1 structure mapping this structure is in GOOINC(SA$BUFHE):

```
/* ============ GSI buffer structure ===========================*/
DCL P_SA$bufhe         POINTER;
DCL 1 SA$bufhe     BASED(P_SA$bufhe),
    2 IA$bufhe_DLEN     BIN FIXED(15), /* Data length            */
    2 IA$bufhe_TLEN     BIN FIXED(15), /* Spare = 0              */
    2 IA$bufhe_TYPE     BIN FIXED(15), /* Type                   */
    2 IA$bufhe_SUBTYPE  BIN FIXED(15), /* Subtype                */
    2 IA$bufhe_USED     BIN FIXED(15), /* Used length            */
    2 HA$bufhe_END      BIN FIXED(7),  /* first buf.el.is fragment*/
    2 HA$bufhe_BEGIN    BIN FIXED(7),  /* last buf.el.is fragment */
    2 LA$bufhe_BUF      BIN FIXED(31), /* Buffer number          */
    2 LA$bufhe_EVT      BIN FIXED(31), /* number of fragments    */
    2 LA$bufhe_CURRENT_I BIN FIXED(31),/* for unpack             */
    2 LA$bufhe_TIME(2)  BIN FIXED(31), /* time stamp             */
    2 LA$bufhe_FREE(4)  BIN FIXED(31), /* Byte order tag         */
                                       /* Length of last event   */
                                       /* free                   */
                                       /* free                   */
    2 IA$bufhe_DATA(1 REFER(IA$bufhe_DLEN))
                        BIN FIXED(15); /* data field             */
/*-------------------------------------------------------------*/
```

F.3.4 Buffer Element Header

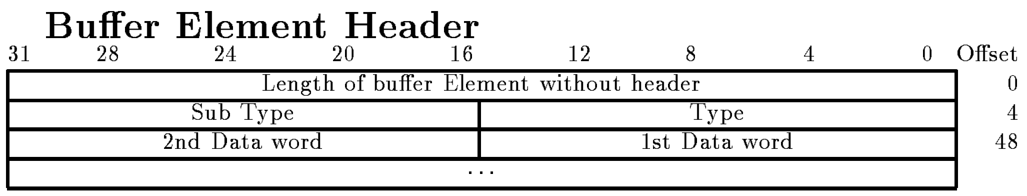


Figure F.4: Buffer Element Header Structure

The total length of the buffer element header is 8 bytes.

Length of buffer element Length of the buffer element without this header in 16–bit words.

(BIN FIXED (31)).

Type	A number specifying the buffer element type. (BIN FIXED (15)).
Sub Type	A number specifying the buffer element subtype. (BIN FIXED (15)).
Data	Any structure of data depending on type and subtype. (any).

Structure Declaration

The PL/1 structure mapping this structure is in GOOINC(SA$EVHE):

```
/* ================ GSI Event header ===================== */
DCL P_SA$evhe       POINTER;
DCL 1 SA$evhe         BASED(P_SA$evhe),
    2 LA$evhe_dlen    BIN FIXED(31), /* data length in words */
    2 IA$evhe_type    BIN FIXED(15), /* type                 */
    2 IA$evhe_subtype BIN FIXED(15), /* subtype              */
    2 IA$evhe_data(1 REFER(IA$evhe_dlen))
                      BIN FIXED(15); /* first data word      */
/*-------------------------------------------------------*/
```

F.3.5 Event Spanning

Events could sometimes be bigger than a buffer. Therefore an event may span over buffer boundaries. The two bits in the buffer header specify if the first or last element in the buffer are fragments. When the last element is a fragment, the length field keeps the length of the fragment. The total length is in the buffer header. The next buffer contains a fragment at the beginning. This fragment is preceeded by an element header (see above). The length field keeps the length of the fragment, type and subtype are the same as for the first fragment.

NOTE Any software processing buffers must be prepared to get buffers with 'lonely' fragments, i.e. at the beginning of a file there might be a fragment. Similar the last buffer in a file may contain a fragment at the end.

F.4 Buffer Types

F.4.1 Overview

Presently the following buffer types and buffer element types are used

2000,1	File header. Buffer header plus one buffer element data.
3000,1	Acknowledge buffer. This buffer contains no data but marks the end of a buffer stream.
1 ,1	MBD buffer. This is a no standard GOOSY header. The buffer must be processed by user written routines.
2 ,1	Buffer contains J11 generated SILENA formatted events with standard header of type:

1 ,1	SILENA formatted subevents.

3 ,1	Buffer contains compressed buffer elements of type:

3 ,1	Compress mode 1
3 ,2	Compress mode 2

4 **,1**	Buffer contains events of type:

4 **,1**	uncompressed events
4 **,2**	compressed events (zeros suppressed)

5 ,1	Buffer contains LRS FERA events with standard header of type:

5 ,1	FERA formatted subevents

6 ,1	Buffer contains standard MBD events of type:

6 ,1	standard events with structure defined by J11 programs.

7 ,s	Buffer contains standard MBD events of type:

7 ,s	events with user structure defined by J11 programs.

The subtype numbers can be specified by the user.

10 ,1	Buffer contains VME formatted events of type 10,1.

10 ,1	standard event written by VME system. Event is composed by subevents of type 10,1 and 10,2.

12 ,1	Buffer contains SILENA formatted events without standard header as stored in FERA memory.
15 ,1	Buffer contains LRS FERA events without standard header as stored in FERA memory.
1000,s	GOOSY Data Element. Type specified by s.

1000,1	GOOSY spectrum
1000,2	GOOSY condition
1000,3	GOOSY picture
1000,4	GOOSY polygon
1000,5	GOOSY calibration
1000,6	GOOSY Data Element (any)

10101,n	External user buffer type (Mainz).
10102,n	External user buffer type (THD).
10103,n	External user buffer type (CAVEB).
any	Any buffer may contain following element types

9000,1	time stamp
2001,1	CAMAC Readout table (initialization)
2001,2	CAMAC Readout table (readout)
2001,3	CAMAC Readout table (reset)
2002,1	Fastbus readout table (init)
2003,1	VME Readout table (init)

F.4.2 File Header Buffer

Figure F.5 shows the GOOSY File header structure. Note, that the File Header Buffer is a standard GOOSY buffer.

Buffer header information:

Length of data field Depends on buffer length.
(BIN FIXED(31))

Type A number specifying the buffer type. For this file header always = 2000.
(BIN FIXED(15))

Subtype A number specifying the buffer subtype. For this file header always = 1.
(BIN FIXED(15))

Used Length of Data Field Depends on length of comment.
(BIN FIXED(15))

Fragment begin This file header buffer contains *never* incomplete buffer elements. This field is always = 0.
(BIN FIXED(7))

Fragment end This file header buffer contains *never* incomplete buffer elements. This field is always = 0.
(BIN FIXED(7))

Number of Buffer Elements For this file header always = 1.
(BIN FIXED(31))

Buffer Number A current number of buffers of the same type.
(BIN FIXED(31))

Current Index A longword not used.
(BIN FIXED (31)).

Time stamp A quadword for the system time in VAX/VMS binary format. This is the number of 100-nanoseconds since 17–Nov–1858 00:00.
(BIT(64)).

4 Free Longwords ((4) BIN FIXED(31)).

File Header Buffer

31 28 24	20 16	12 8 4 0	Offset
Length of buffer without header			0
Buffer Subtype = 1		Buffer Type = 2000	4
Fragment begin= 0	Fragment end= 0	Used Length of Data field = 1000	8
Buffer Number for this Type - Sub Type			12
Number of Buffer Elements or Fragments of Buffer Elements = 1			16
Not used			20
Time stamp VMS 64 bit format			24
Time stamp VMS 64 bit format			28
4 Longwords reserved			32
. . .			
Tape label(30 char.)		Used length of tape label	48
Tape label continuation			
. . .			
File name (86 char.)		Used length of File name	80
File name continuation			
. . .			
User name (30 char.)		Used length of user name	168
User name continuation			
. . .			
Date "`dd-mmm-yyyy hh:mm:ss.mm`" (24 character)			200
Date continuation			
Run ID (66 char.)		Used length of Run ID	224
Run ID continuation			
. . .			
Experiment (66 char.)		Used length of Experiment	292
Experiment continuation			
. . .			
Number of Lines = n			360
Line 1 (78 char.)		Used length of Line 1	364
Line 1 continuation			
Line 2 (78 char.)		Used length of Line 2	
Line 2 continuation			
. . .			
Line n (78 char.)		Used length of Line n	
Line n continuation			

Figure F.5: File Header Structure

File header specific Information:

Used Length of Tape Label Number of characters used in the next field.
(BIN FIXED(15)).

Tape Label Contains the tape label of the ANSI tape, if the file was created on a tape.
(CHAR(30) VAR).

Used Length of File name Number of characters used in the next field.
(BIN FIXED(15)).

File name Name of file at the time of creation. The used Length is specified by the "Used Length of File name" field. If one wants to send the output files to the IBM, the filenames must follow some conventions:

1. Maximal length 25 char (including type)
2. Maximal 8 char or 7 digits between two underscores (No $).
3. File type must be .LMD

(CHAR(86) VAR).

Used Length of User name Number of characters used in the next field.
(BIN FIXED(15)).

User name User name of the creating VAX/VMS process.
(CHAR(30) VAR).

Date Character string of the creation date in the format "`dd-mmm-yyyy hh:mm:ss.mm` " where `dd` is the day of month, `mmm` is the 3 character abbreviation of the english spelled month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC), and `yyyy` is the year, `hh` are hours, `mm` minutes, `ss.mm` are seconds, e.g. "21-OCT-1986 14:34:30.10 ". This date string is always padded by a space character.
(CHARACTER(24)).

Used Length of Run Identification Number of characters used in the next field.
(BIN FIXED(15)).

Run Identification Character string to identify the experiment run corresponding to this file. The contents is user defined. The string can have a maximum of 66 characters. The actual length is defined by the "Length of Run Identification" field.
(CHARACTER(66) VAR).

Used Length of Experiment Name	Number of characters used in the next field. (BIN FIXED(15)).
Experiment Name	Character string to identify the experiment corresponding to this file. The contents is user defined. The string can have a maximum of 66 characters. The actual length is defined by the "Length of Experimenter Name" field. (CHARACTER(66) VAR).
Number of Lines	Number of 78–character lines following. (BIN FIXED(31)).
Used Length of line	Number of characters used in the next field. (BIN FIXED(15)).
Comment Lines	Character string array to characterize the contents of this file. The lines are user defined. The header can have a maximum of 46 lines. The actual number of lines is defined by the "Number of Lines" field. ((*) CHARACTER(78) VAR).

Structure Declaration

The file header buffer is mapped by the PL/1 structure GOOINC(SA$FILHE):

```
/* ============= GSI file header buffer structure ================*/
DCL L_SA$filhe_lines    BIN FIXED(31);/* number of lines         */
DCL P_SA$filhe          POINTER;
DCL 1 SA$filhe      BASED(P_SA$filhe),
    2 IA$filhe_DLEN     BIN FIXED(15), /* Data length            */
    2 IA$filhe_TLEN     BIN FIXED(15), /* Total length           */
    2 IA$filhe_TYPE     BIN FIXED(15), /* Type                   */
    2 IA$filhe_SUBTYPE  BIN FIXED(15), /* Subtype                */
    2 IA$filhe_USED     BIN FIXED(15), /* Used length            */
    2 HA$filhe_END      BIN FIXED(7),  /* first event is fragment */
    2 HA$filhe_BEGIN    BIN FIXED(7),  /* last event is fragment  */
    2 LA$filhe_BUF      BIN FIXED(31), /* Buffer number          */
    2 LA$filhe_EVT      BIN FIXED(31), /* number of fragments    */
    2 LA$filhe_CURRENT_I BIN FIXED(31),/* for unpack             */
    2 LA$filhe_TIME(2)  BIN FIXED(31), /* time stamp             */
    2 LA$filhe_FREE(4)  BIN FIXED(31), /* free                   */
    2 CA$filhe_label    CHAR(30) VAR,  /* tape label             */
    2 CVA$filhe_file    CHAR(86) VAR,  /* file name              */
    2 CA$filhe_user     CHAR(30) VAR,  /* user name              */
    2 CA$filhe_time     CHAR(24),      /* time and date          */
    2 CVA$filhe_run     CHAR(66) VAR,  /* run id                 */
    2 CVA$filhe_exp     CHAR(66) VAR,  /* experiment             */
    2 LA$filhe_lines    BIN FIXED(31), /* number of lines        */
    2 CVA$filhe_line(L_SA$filhe_lines REFER(LA$filhe_lines))
                        CHAR(78) VAR;  /* comment lines          */
/*---------------------------------------------------------------*/
```

F.4.3 GOOSY Data Element Buffers

These buffers of type 1000 contain GOOSY Data Elements. These are encoded in special structures. The subtype may be used to select different Data Element types. (Not yet impl.)

F.4.4 GOOSY Listmode Data Buffers

Listmode data buffers contain buffer elements called *events*. The different event types are described in the next section.

F.5 Event Structures

F.5.1 Event Type 3 (compressed)

Figure F.6 shows the event structure of type 3. Behind the header there follows one Data Element which is compressed. Two compress modes are supported. One adds a BIT(32) longword for each 32 Longwords. Zero longwords are suppressed and marked in the bitstring. The other adds counter longwords containing the number of following zero or nonzero longwords. **These buffer elements are longword aligned!**

Event Type 3

31 28 24 20 16	12 8 4 0	Offset
Data length		0
Subtype = 1, 2	Type = 3	4
Compression mode		8
Length of uncompressed data		12
First compressed longword		16
...		

Figure F.6: Event structure type 3 (compressed)

Compression mode Two modes are provided: Bit mask mode and counter mode. (BIN FIXED (31)).

Length of uncompressed data Length of the original Data Element. (BIN FIXED (31)).

Usage

The analysis program can output Data Elements event by event. These Data Elements are copied to GOOSY buffers. Two storage modes can be selected: Compress and Copy mode. With compress mode the above structure is copied to the buffer. The original structure of the Data Element is lost. If the buffer is input by another analysis, the compressed buffer element is decompressed and restored. The advantage is that arbitrary data structures can be compressed, the disadvantage, that the compress/decompress procedure is time consuming. The pack routine is X$PACMP, the unpack routine X$UPCMP.

F.5.2 Event Type 4

Event Type 4, Subtype 1 (block)

Figure F.7 shows the event structure of type 4. Behind the header one Data Element follows. The structure is processed as a word array. **These buffer elements are NOT longword aligned!**

Event Type 4, Subtype 1

31 28 24 20 16	12 8 4 0	Offset
Data length		0
Subtype = 1	Type = 4	4
second data word	First data word	8
. . .		

Figure F.7: Event structure type 4, subtype 1 (block)

Event Type 4, Subtype 2 (no zero's)

Figure F.8 shows the event structure of type 4. Behind the header one Data Element follows. The structure is processed as a word array. Each data word is specified by an identification number, e.g. an ADC number. **These buffer elements are longword aligned!**

Event Type 4, subtype 2

31 28 24 20 16	12 8 4 0	Offset
Data length		0
Subtype = 2	Type = 4	4
First data word	First data word id	8
Second data word	Second data word id	12
. . .		

Figure F.8: Event structure type 4, subtype 2 (no zeros's)

Structure Declaration

Both event structures are copied to a Data Element in the Data Base with structure GOOTYP(SA$EVENT):

```
/* ================= GSI Event header 4,1 =====================*/
DCL P_SA$event       POINTER;
DCL 1 SA$event         BASED(P_SA$event),
    2 IA$event_dlen    BIN FIXED(15),  /* data length in words */
    2 IA$event_tlen    BIN FIXED(15),  /* not used =0          */
    2 IA$event_type    BIN FIXED(15),  /* type = 4             */
    2 IA$event_subtype BIN FIXED(15),  /* subtype = 1          */
    2 IA$event(512)    BIN FIXED(15);  /* data.                */
/*------------------------------------------------------------*/
```

Note that this structure contains no REFER because it is used to create the event Data Element in the Data Base. For special purposes the user may create his own event structure. The first four words must be declared as shown above.

Usage

The analysis program can output Data Elements event by event. These Data Elements are copied to GOOSY buffers. Two storage modes can be selected: Compress and Copy mode. With copy mode the above structure (subtype 1) is copied to the buffer. The original structure of the Data Element is lost. If the buffer is input by another analysis, the buffer element is copied back to the Data Element. The advantage is that arbitrary data structures can be copied, the disadvantage that no compression is done. The original Data Element must have a standard header.

Both formats are also used by the CAMAC single crate system controlled by a J11. The zero suppression can be enabled during data acquisition. The unpack routine for these events is X$UPEVT.

F.5.3 Event Type 1 (Buffer Type 2, SILENA)

These events have a standard buffer element header. This header must be generated by the processor reading out the ADC. Otherwise these events are stored without header in buffers of type 12. As shown in figure F.9, the buffer element is composed of a header followed by several Data Elements. These Data Elements are produced by the ADC/TDC modules type SILENA 4418x. **These buffer elements are NOT longword aligned!**

Event Type 1

31 28 24 20 16		12 8	4 0	Offset
Data length				0
Subtype = 1	Type = 1			4
Pattern Word	1	No. of Data Words	Subevent Id.	8
2nd Data Word	1st Data Word			12
. . .				
Pattern Word	1	No. of Data Words	Subevent Id.	
2nd Data Word	1st Data Word			
. . .				

Figure F.9: Event structure type 1 (SILENA)

Sub Event Id A number from 0 to 127 defining the sub event to which the following pattern word belongs. **This byte is NOT longword aligned!**

(BIN FIXED (7)).

Number of Data Words The number of data words (e.g. ADC values) following the pattern word. This number must be identical to the number of bits set in the pattern word.

(BIN FIXED (7) but with the highest bit, the event tag bit set to 0 !!).

Event Tag Bit The bit 2^{15} marks the event tag word. This bit is set in subevent header longwords.

(BIT (1)).

Pattern Word Each bit in the pattern word corresponds to a data word (e.g. ADC value) following this pattern word. The bit 2^{0} corresponds to the first word. The number of bits set in the pattern word must be identical to the "Number of Data Words" field of this Simple Event Structure.

(BIT (16)).

Data Words The number of 16 bit data words (e.g. ADC data) is defined by the number of bits set in the pattern word or the identical "Number of Data Words" field in the structure.

((n) BIN FIXED (15)).

Usage

This format is presently not used.

F.5.4 Event Type 5 (LRS FERA)

These events have a standard buffer element header. This header must be generated by the processor reading out the ADC. Otherwise these events are stored without header in buffers of type 15. As shown in figure F.10, the buffer element is composed of a header followed by several Data Elements. These Data Elements are produced by the ADC/TDC modules type LRS 4300 (FERA). **These buffer elements are NOT longword aligned!**

Subevent Id A number defining the sub event to which the following subevent belongs. **This byte is NOT longword aligned!**

(BIN FIXED (7)).

# Data Words The number of data words (e.g. ADC values) following the pattern word.

(BIN FIXED (7) but with the highest bit, the event tag bit set to 0 !!).

Event Tag Bit The bit 2^{15} marks the event tag word. This bit is set in subevent header longwords.

(BIT (1)).

Event Type 5

31	28	24 20	16	12	8			4 0	Offset
Data length									0
Subtype = 1			Type = 5						4
0	1st SA	1st data word	1	# Data Words	0	0	0	Subevent Id	8
0	3rd SA	3rd Data Word	0	2nd SA	2nd Data Word				12
. . .									
0	1st SA	1st data word	1	# Data Words	0	0	0	Subevent Id	
0	3rd SA	3rd Data Word	0	2nd SA	2nd Data Word				
. . .									

Figure F.10: Event structure type 5 (LRS FERA)

Data Words The number of 11 bit data words (e.g. ADC data) is defined by "number of data words". The source of the data words is specified by the "SA" field.

(BIT(11)).

SA Subaddress of the source of the data word.

(BIT(4)).

Usage

This format is presently not used.

F.5.5 Event Type 6 (MBD buffer type 6)

Figure F.11 shows the event structure of type 6 in buffers of type 6. The subevent structure is produced by the J11 and MBD programs. **These buffer elements are NOT longword aligned!**

Subevent length Length of subevent in words **excluding** header longword. **This word is NOT longword aligned!**

(BIN FIXED (15)).

CAMAC crate The number of the CAMAC crate where the subsequent subevent data came from.

(BIN FIXED (7))

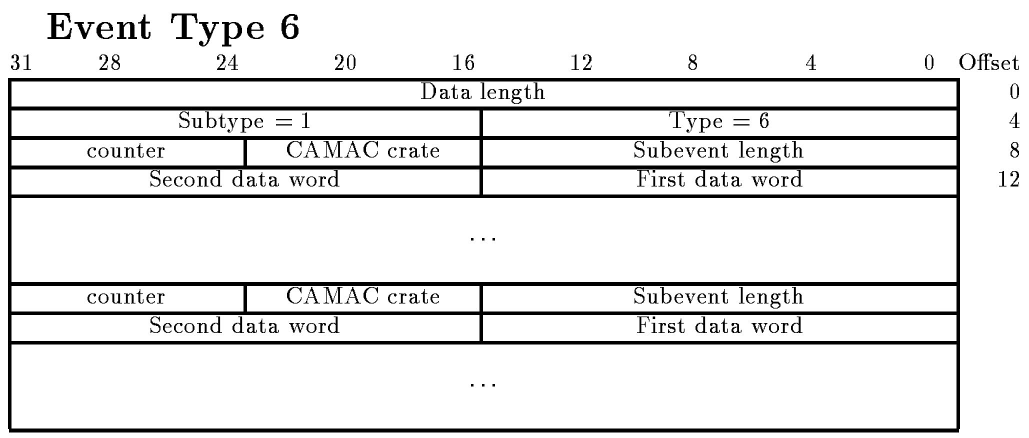


Figure F.11: Event structure type 6 (MBD buffer type 6)

Counter	A counter to check correct order of events and subevents. (BIN FIXED (7))
Data Words	Data. (BIN FIXED (15))

Structure Declarations

The subevent structure is mapped by the PL/1 structure GOOINC(SA$ME6_1):

```
/*======== Declaration of MBD event structure 6,1 ===========*/
  DCL P_SA$ME6_1 POINTER INIT(NULL);
  DCL 1 SA$ME6_1 BASED(P_SA$ME6_1),
 2 IA$ME6_1_slen    BIN FIXED(15), /* subevent length */
 2 HA$ME6_1_crate   BIN FIXED(7),  /* crate           */
 2 HA$ME6_1_event   BIN FIXED(7),  /* event count     */
 2 IA$ME6_1_data(IA$ME6_1_slen)
      BIN FIXED(15), /* data words      */
 2 SA$ME6_1_next,
   3 IA$ME6_1_nslen   BIN FIXED(15),
   3 HA$ME6_1_nscrate BIN FIXED(7);
/*-------------------------------------------------------------*/
```

The event structure is copied to a Data Element in the Data Base with structure GOOTYP(SA$MBD):

```
/* ===== Declaration of MBD event structure 6,1 ===== */
  DCL P_SA$MBD POINTER;
  DCL 1 SA$MBD BASED(P_SA$MBD),
2 IA$MBD_dlen    BIN FIXED(15),
2 IA$MBD_tlen    BIN FIXED(15),
2 IA$MBD_type    BIN FIXED(15),
2 IA$MBD_subtype   BIN FIXED(15),

        2 SA$MBD_C1,
3 IA$MBD_C1_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C1(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C2,
3 IA$MBD_C2_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C2(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C3,
3 IA$MBD_C3_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C3(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C4,
3 IA$MBD_C4_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C4(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C5,
3 IA$MBD_C5_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C5(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C6,
3 IA$MBD_C6_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C6(99)    BIN FIXED(15), /* data words */
        2 SA$MBD_C7,
3 IA$MBD_C7_slen   BIN FIXED(15), /* subevent length    */
3 IA$MBD_C7(99)    BIN FIXED(15); /* data words */
```

Note that this structure contains no REFER because it is used to create the event Data Element in the Data Base. For special purposes the user may create his own event structure. The first four words must be declared as shown above. If the length of the subcrate structures are different, a special unpack routine must be provided.

Usage

This will be the standard MBD event structure. The event Data Element with the structure SA$MBD will be filled by a standard unpack routine X$UPMBD.

F.5.6 Event Type 7 (MBD buffer type 7)

Figure F.12 shows the event structure of type 7 in buffers type 7. The subevent structure is provided by the user. **These buffer elements are NOT longword aligned!**

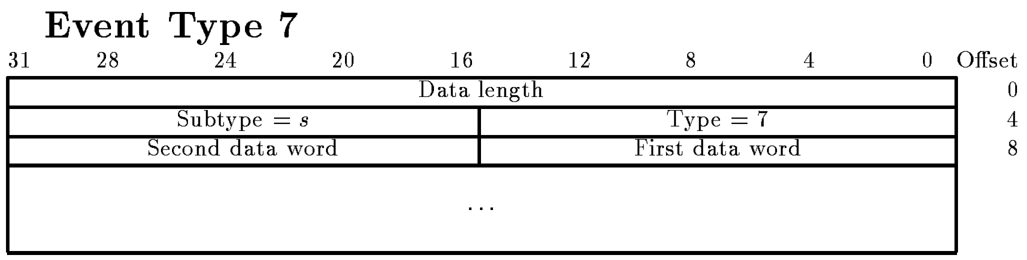


Figure F.12: Event structure type 7 (MBD buffer type 7)

Type Must be 7.
(BIN FIXED (15)).

Subtype The subtype can be specified by the user. The buffer subtype must be equal to this event subtype.
(BIN FIXED (15)).

Data Words Data. Contains subevent structures defined by the user. Different structures can be marked by different type/subtype numbers.
(BIN FIXED (15))

Usage

This type allows users to write specific applications requiring specific event structures.

F.5.7 Event Type 10 (VME)

This structure is composed by the EB. It is mapped by SA$VE10_1 in library GOOINC.

```
/* ================ GSI VME Event header ====================== */
DCL P_SA$ve10_1       POINTER;
DCL 1 SA$ve10_1         BASED(P_SA$ve10_1),
    2 LA$ve10_1_dlen    BIN FIXED(31),
    2 IA$ve10_1_type    BIN FIXED(15),
    2 IA$ve10_1_subtype BIN FIXED(15),
    2 IA$ve10_1_dummy   BIN FIXED(15),
```

Event Type 10, Subtype 1

31 28 24 20 16	12 8 4 0	Offset
Data length [words]		0
Subtype = 1	Type = 10	4
Trigger	Not used	8
Event counter		12
Subevent 1		16
...		
Subevent n		16+x

Figure F.13: Event Structure

```
    2 IA$ve10_1_trigger BIN FIXED(15),
    2 LA$ve10_1_count   BIN FIXED(31),
    2 IA$ve10_1(LA$ve10_1_dlen-4)    BIN FIXED(15),
    2 LA$ve10_1_next    BIN FIXED(31);
/*-------------------------------------------------------------------*/
```

CAMAC Subevent Structure 10,1

This subevent structure is written by the ROP or the FEP. It is defined in SA$VES10_1 in library GOOINC.

Subevent Type 10, Subtype 1

31 28 24	20 16	12 8 4 0	Offset
Subevent Data length [words]			0
Subevent subtype = 1		Subevent type = 10	4
Control	subcrate	Processor ID	8
CAMAC value		CAMAC module ID	12
...			

Figure F.14: CAMAC Subevent Structure

```
/* ================= GSI VME Subevent header ====================== */
DCL P_SA$ves10_1       POINTER;
DCL 1 SA$ves10_1         BASED(P_SA$ves10_1),
    2 LA$ves10_1_dlen    BIN FIXED(31),
    2 IA$ves10_1_type    BIN FIXED(15),
    2 IA$ves10_1_subtype BIN FIXED(15),
    2 IA$ves10_1_procid  BIN FIXED(15),
    2 HA$ves10_1_subcrate BIN FIXED(7),
    2 HA$ves10_1_control BIN FIXED(7),
    2 IA$ves10_1(LA$ves10_1_dlen-2)    BIN FIXED(15),
    2 LA$ves10_1_next    BIN FIXED(31);
/*-------------------------------------------------------------------*/
```

FASTBUS Subevent Structure 10,2

This subevents are written from the AEB. The header structure is defined in SA$VES10_1 in library GOOINC.

Subevent Type 10, Subtype 2

31 28	24 20	16 12 8 4 0	Offset
Subevent Data length [words]			0
Subevent subtype = 2		Subevent type = 10	4
Control	subcrate	Processor ID	8
Fastbus module header			16
...			

Figure F.15: Fastbus Subevent Structure

The following structure maps to the data field. It is defined in SA$vesfb in library GOOINC.

Fastbus module header

31 28	24 20	16 12 8 4 0	Offset
Longwords	Geo.addr.	Module ID	0
1st data word			4
...			

Figure F.16: Fastbus Module header

```
/* Fastbus module header maps to IA$ves10_2(i) */
DCL P_SA$vesfb       POINTER;
DCL 1 SA$vesfb BASED(P_SA$ves_fb),
    2 IA$vesfb_id BIN FIXED(15),
    2 HA$vesfb_addr BIN FIXED(7),
    2 HA$vesfb_lwords BIN FIXED(7),
    2 LA$vesfb_data(HA$vesfb_lwords)    BIN FIXED(31),
    2 LA$vesfb_next    BIN FIXED(31);
```

Fastbus data word

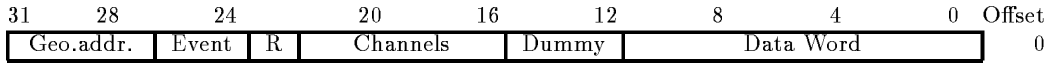


Figure F.17: Fastbus Data Word

One data word looks like

```
/* Structure of data words */
/* Numbers from 1 to 32 can be used in POSINT */
%REPLACE FBDATA_d    BY 1;   %REPLACE FBDATA_d_l  BY 12;
%REPLACE FBDATA_x    BY 13;  %REPLACE FBDATA_x_l  BY 4;
%REPLACE FBDATA_ch   BY 17;  %REPLACE FBDATA_ch_l BY 7;
%REPLACE FBDATA_r    BY 24;  %REPLACE FBDATA_r_l  BY 1;
%REPLACE FBDATA_ev   BY 25;  %REPLACE FBDATA_ev_l BY 3;
%REPLACE FBDATA_ad   BY 28;  %REPLACE FBDATA_ad_l BY 5;
DCL P_SI$FBDATA POINTER; /* maps to LA$vesfb_data(i) */
DCL 1 SI$FBDATA BASED(P_SI$FBDATA),
    2 BI$FBDATA_d  BIT(12) /* data word */
    2 BI$FBDATA_x  BIT(4), /* dummy     */
    2 BI$FBDATA_ch BIT(7), /* channel   */
    2 BI$FBDATA_r  BIT(1), /* range     */
    2 BI$FBDATA_ev BIT(3), /* event     */
    2 BI$FBDATA_ad BIT(5); /* geo addr. */
/*-------------------------------------------------------------------*/
```

F.6 Buffer Element Structures

F.6.1 Buffer Element Type 9000 (Time Stamp)

Figure F.18 shows the Time Stamp structure (buffer element structure type 9000). **These buffer elements are longword aligned!**

Buffer Element Type 9000, Time Stamp

31 28 24 20 16	12 8 4 0	Offset
Data length=10		0
Subtype = 0	Type = 9000	4
Date "`dd-mmm-yyyy hh:mm:ss.mm`" (24 character)		8
Date continuation		

Figure F.18: Buffer Element structure type 9000, Time Stamp

Date Character string of the creation date in the format "`dd-mmm-yyyy hh:mm:ss.mm` " where `dd` is the day of month, `mmm` is the 3 character abbreviation of the english spelled month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC), and `yyyy` is the year, `hh` are hours, `mm` minutes, `ss.mm` are seconds, e.g. "21-OCT-1986 14:34:30.10 ". This date string is always padded by a space character. (CHARACTER(24)).

Usage

Not yet used.

F.6.2 Buffer with GOOSY Data Elements

Buffer type 1000 contains GOOSY Data Elements. The subtype specifies the kind of Data Element.

GOOSY spectrum

GOOSY condition

GOOSY picture

GOOSY polygon

GOOSY calibration

GOOSY Data Element

F.7 Nonstandard Buffer Structures

F.7.1 Buffer Type 12 (SILENA)

Figure F.19 shows the subevent structure as produced by one ADC/TDC module of type SILENA 4418x. Several modules produce several subsequent structures. **These buffer elements are NOT longword aligned!**

SILENA Data Structure

31 28 24 20 16		12 8	4 0	Offset
Pattern Word	1	No. of Data Words	Subevent Id.	0
2nd Data Word	1st Data Word			4
. . .				

Figure F.19: Data structure SILENA ADC

Subevent Id A number from 0 to 127 defining the subevent to which the following pattern word belongs. **This byte is NOT longword aligned!**

(BIN FIXED (7)).

Number of Data Words The number of data words (e.g. ADC values) following the pattern word. This number must be identical to the number of bits set in the pattern word.

(BIN FIXED (7) but with the highest bit, the event tag bit set to 0 !!).

Event Tag Bit The bit 2^{15} marks the event tag word. This bit is set in the subevent header longword.

(BIT (1)).

Pattern Word Each bit in the pattern word corresponds to a data word (e.g. ADC value) following this pattern word. The bit 2^0 corresponds to the first word. The number of bits set in the pattern word must be identical to the "Number of Data Words" field of this Simple Event Structure.

(BIT (16)).

Data Words The number of 16 bit data words (e.g. ADC data) is defined by the number of bits set in the pattern word or the identical "Number of Data Words" field in the structure.

((n) BIN FIXED (15)).

Usage

Not yet used.

F.7.2 Buffer Type 15 (LRS FERA)

Figure F.20 shows the subevent structure as produced by one ADC/TDC module of type LRS 4300 (FERA). Several modules produce several subsequent structures. **These buffer elements are NOT longword aligned!**

LRS 4300 (FERA)

31	28	24 20	16	12		8		4 0	Offset
0	1st SA	1st data word	1	# Data Words	0	0	0	Subevent Id	0
0	3rd SA	3rd Data Word	0	2nd SA	2nd Data Word				4
				...					
0	last SA	last Data Word	0	··· SA	··· Data Word				

Figure F.20: Data structure LRS FERA

Subevent Id	A number defining the subevent to which the following subevent belongs. **This byte is NOT longword aligned!** (BIN FIXED (7)).
# Data Words	The number of data words (e.g. ADC values) following the pattern word. (BIN FIXED (7) but with the highest bit, the event tag bit set to 0 !!).
Event Tag Bit	The bit 2^{15} marks the event tag word. This bit is set in the subevent header longword. (BIT (1)).
Data Words	The number of 11 bit data words (e.g. ADC data) is defined by "number of data words". The source of the data words is specified by the "SA" field. (BIT(11)).
SA	Subaddress of the source of the data word. (BIT(4)).

Usage

Not yet used.

Appendix G

GOOSY Error Recovery

GOOCMD

Prefix	XCMD_
Object name	GOOMSGLIB(XCMD)
File name	XCMD.MSG
Dataset	RZ88.XCMD.MSG
Identifier	V01.00A
Facility	101

ABRMAT

MESSAGE	Parser, full match for abriviated token
RECOVERY	No recovery required. Signals, that a token or tag searched in a command line or a CDL descriptor list has been found in its minimum abriviated form.
SEVERITY	SU
Author	Walter F.J. Mueller

AMBMAT

MESSAGE	'<token>' is ambiguous. Reenter command fully specified.
RECOVERY	The given command contains at least one abriviated token which can not be located unambiguously. Reenter the command with full specified tokens.
SEVERITY	ER
Author	Walter F.J. Mueller

AMBPAR

MESSAGE	'<token>' is ambiguous. Reenter parameter name.
RECOVERY	The given command contains at least one abriviated parameter name which can not be located unanbiguously. Reenter the command with fully specified parameters.
SEVERITY	ER
Author	Walter F.J. Mueller

AMBTYP

MESSAGE	Ambiguous process types, specify process type explicitely
RECOVERY	The given command is available in the context of more than one process. It is nessecarry to specify the process type prefix for those commands.
SEVERITY	ER
Author	Walter F.J. Mueller

BADERRSTACK

MESSAGE	Bad error stack, messages found but success exit status
RECOVERY	Indicates, that a command action routine exits with a success status but with messages on the error stack. This was probably caused by a faulty error handling. NOTE, that the handling of an error should include the print or deletion of it's assiociated error messages in the error stack. Check which module issued the messages and where the exit status was changed. Clean up error handling.
SEVERITY	WA
Author	Walter F.J. Mueller

BREAK

MESSAGE	Interrupted command

RECOVERY	none
SEVERITY	WARNING
Author	H.G.Essel

BUGCHK

MESSAGE	BUGCHECK. Internal consistency check, syndrome: <syndrome>
RECOVERY	Signals, that a procedure detected a fatal internal consistency error which is unextected and unrecoverable. The syndrome text specifies the bugcheck reason, the interpretation depends on the procedure. Ask the author to correct the problem.
SEVERITY	ER
Author	Walter F.J. Mueller

CMDNOTFND

MESSAGE	No command found - use $ RECALL /ALL to list saved commands
RECOVERY	A $ RECALL command failed to find the desired command line. Either the command number out of range or there is no command beginning with the specified leading substring. Use either $ RECALL/ALL to list all saved commands or a plain $ RECALL command to enter the screen mode. There must be at least one command on recall stack.
SEVERITY	WA
Author	Walter F.J. Mueller

CNVARR

MESSAGE	Array dimension of <dimension> is to small for <string>
RECOVERY	Signals, that the array of the size <dimension> passed to a prompting routine is to small to receive all values which result from the conversion of the <string>. Either enter less values for that parameter or increase the size of the array in the calling procedure.

SEVERITY	ER
Author	Walter F.J. Mueller

CNVERR

MESSAGE	Conversion error in field <field> for prompt <prompt>
RECOVERY	:BR. Reenter the correct value
SEVERITY	ER
Author	Walter F.J. Mueller

CNVLIM

MESSAGE	Conversion outside limits '<lowerbound upperbound>'
RECOVERY	Signals, that a parameter value is outside the limits specified with the MIN and MAX tags of the command description. Reenter a value in the given limits.
SEVERITY	ER
Author	Walter F.J. Mueller

CONFLPARA

MESSAGE	Conflicting parameters: <reason>
RECOVERY	Some parameters specified for a command conflict with each other. The <reason> string should describe the problem in more detail. Look in the command description and respecify the command.
SEVERITY	ER
Author	Walter F.J. Mueller

CTIMEOUT

MESSAGE	Interrupt by timeout.

RECOVERY The menu input is terminated after n seconds when no input is seen.

SEVERITY WA

Author G.Schneider

DUMMY

MESSAGE General purpose dummy procedure executed

RECOVERY No recovery required. Signals, that the general purpose dummy C$DUMMY of the command subsystem has been executed. This may be used by the calling procedure to check, whether the dummy has been used or a real action occured.

SEVERITY SU

Author Walter F.J. Mueller

ERRCDL

MESSAGE Error in command definition
verb: <verb CDL>
params: <parameter CDL>

RECOVERY Signals that the creation of the command <verb CDL> with the parameter description <parameter CDL> results in an error. In general there are other messages on the message stack which explain the source of the error.
Correct the problem and reenter the command definition.

SEVERITY WA

Author Walter F.J. Mueller

ERRCMD

MESSAGE Error in command line:
<line>

RECOVERY Signals that the execution of the command <line> results in an error. In general there are other messages on the message stack which explain the source of the error.
Correct the problem and reenter the command line.

SEVERITY WA

Author Walter F.J. Mueller

ERROR

MESSAGE Error condition: <syndrome>

RECOVERY No recovery possible. Signals, that a general error condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.

SEVERITY ER

Author Walter F.J. Mueller

EXEABORT

MESSAGE Abort of command procedure due to an error at line
<line>

RECOVERY A command in a command procedure returned a nonsuccess status and resulted in the abortion of the procedure.
Fix the errorous command and retry the procedure.

SEVERITY ER

Author Walter F.J. Mueller

EXEFNF

MESSAGE Command procedure file 'file_name' not found

RECOVERY A command procedure could not be executed because the procedure file could not be opened.
Fix the errorous file name in the $ EXECUTE or @ command and retry.

SEVERITY ER

Author Walter F.J. Mueller

EXENEST

MESSAGE	Nesting depth of command procedures exceeded
RECOVERY	Signals, that the maximum command procedure nesting depth of 4 has been exceeded and that the execution of the procedures will be aborted. Reduce the nesting depth or your procedure calls.
SEVERITY	ER
Author	Walter F.J. Mueller

FATAL

MESSAGE	Fatal condition: <syndrome>
RECOVERY	No recovery possible. Signals, that a general fatal condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	FA
Author	Walter F.J. Mueller

FULMAT

MESSAGE	Parser, full match for token
RECOVERY	No recovery required. Signals, that a token or tag searched in a command line or a CDL descriptor list has been found in its unabriviated form.
SEVERITY	SU
Author	Walter F.J. Mueller

GENMAT

MESSAGE	Parser, generic match for token

RECOVERY	No recovery required. Signals, that a token or tag searched in a command line or a CDL descriptor list has been found, but that the found string is an abriviation and longer than the minimum abriviation length.
SEVERITY	SU
Author	Walter F.J. Mueller

ILLCDK

MESSAGE	Illegal CDL tag <tag> for parameter <para>
RECOVERY	Signals that the CDL tag <tag> is either unknown or illegal in this context for the parameter <para>. Correct the error in the CDL definition.
SEVERITY	WA
Author	Walter F.J. Mueller

ILLCDV

MESSAGE	Illegal CDL tag value <value> for parameter <para>
RECOVERY	Signals that the CDL tag value <value> is either out of range or not allowed at all for the parameter <para>. Correct the error in the CDL definition.
SEVERITY	WA
Author	Walter F.J. Mueller

ILLCOMKEY

MESSAGE	Illegal command key word.
RECOVERY	One of the command keywords are illegal due to a wrong character. Reenter correct command.
SEVERITY	ERROR
Author	H.G.Essel

ILLMODE

MESSAGE	The procedure has been called with an illegal mode of '<mode>'
RECOVERY	Signals, that the call of the procedure passed a mode parameter with an illegal value. Check what mode was intended and fix the bug in the CALLING program.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLPAR

MESSAGE	Unknown named parameter or qualifier '<field>'
RECOVERY	The given parameter of qualifier is illegal in this context. Look in the command description for the parameter names and the allowed qualifiers. Reenter the corrected command.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLVALUE

MESSAGE	Illegal qualifier value in field '<field>'
RECOVERY	A qualifier has been specified with a value where none is allowed. Reenter the corrected command.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLVERB

MESSAGE	Illegal command tokens, there is already a namelike command

RECOVERY	It has been attempted to declare two commands with either the same name or with one command being a subcommand of the other. Rename one command.
SEVERITY	ER
Author	Walter F.J. Mueller

INACCOM

MESSAGE	Inaccessible command (previous CDL syntax error)
RECOVERY	The given command cannot be executed because the parameter CDL had a syntax or constency error and could not be parsed. NOTE, that a more detailed error message describing the CDL problem is printed only for the first attempt to execute a command. Correct the CDL, recompile, link ..... (sorry).
SEVERITY	ER
Author	Walter F.J. Mueller

INCMAT

MESSAGE	Parser, incomplete match, '<tokens>' has subcommands
RECOVERY	Signals, that the command token(s) <tokens> are legal but don't specify a command. In other words: <tokens> has still subcommands. Reenter the command with all subcommand tokens.
SEVERITY	ER
Author	Walter F.J. Mueller

INDERR

MESSAGE	Error while executing command procedure
RECOVERY	Signals, that the execution of a command procedure has been aborted because a command ended with an error status. Look for the messages before, which indicates the command in error and correct that command.

SEVERITY	ER
Author	Walter F.J. Mueller

INFORM

MESSAGE	Informational condition: <syndrome>
RECOVERY	No recovery required. Signals, that a general informative condition occured during the execution of the module.
SEVERITY	IN
Author	Walter F.J. Mueller

NEGMAT

MESSAGE	Parser, match for negated qualifier
RECOVERY	No recovery required. Signals, that a negated qualifier has been found in a command line.
SEVERITY	SU
Author	Walter F.J. Mueller

NOINIT

MESSAGE	The command dispatcher is not yet initialized
RECOVERY	A command dispatcher procedure has been executed prior to a call of C$CRECM or C$INIT which perform the initialization of the internal database. Fix the calling program, add a call of C$INIT before the errorous call.
SEVERITY	ER
Author	Walter F.J. Mueller

NOMAT

MESSAGE	Unknown command keyword '<token>'. Reenter corrected command.
RECOVERY	The given command contains at least one unknown token. Reenter the correct command.

SEVERITY ER

Author Walter F.J. Mueller

NULLKEY

MESSAGE Zero length keyword detected in field '<field>'

RECOVERY This status signals, that a keyword of zero length was detected in a command or a command definition. This is in general the result of a single "' or consecutive slashes '//'. Bad commands are for example

```
TEST =
TEST //LOG
```

SEVERITY ER

Author Walter F.J. Mueller

PARSEQ

MESSAGE Illegal : positional parameter behind named parameter

RECOVERY The given command contains a positional parameter after a named parameter, which is not allowed.
Reenter the correct command.

SEVERITY ER

Author Walter F.J. Mueller

PRMABO

MESSAGE Prompt request aborted

RECOVERY Signals that the a prompt request was aborted by entering ^ Z. This signals in general, that the prompt was unexpected and that the was not reasonable value to enter.

SEVERITY WA

Author Walter F.J. Mueller

REQMIS

MESSAGE Required parameter '<param>' is missing

RECOVERY Signals, that a required parameter of a command has been omitted and that the prompting of required parameters is disabled (e.g. in batch jobs or command procedures).
Reenter the command and specify the indicated required parameter.

SEVERITY ER

Author Walter F.J. Mueller

SUBNEST

MESSAGE Parameter substitution nesting exceeded

RECOVERY The parameter substitution ended in an potentially infinite loop. Check whether the formal parameter contains again a substitution character. Correct this and reenter the command line.

SEVERITY WA

Author Walter F.J. Mueller

SUCCESS

MESSAGE Success condition: <syndrome>

RECOVERY No recovery required. Signals only, that a module of the command subsystem completed successfully.

SEVERITY SU

Author Walter F.J. Mueller

UNKNOWNTYPE

MESSAGE Unknown process type in a command type prefix

RECOVERY The process type specified in a command of the format

```
type>command_line
```

is unknown in the current session.
Either reenter the command with another process type if this will support the command or start the want process with a CREATE PROCESS command.

SEVERITY	ER
Author	Walter F.J. Mueller

WARCMD

MESSAGE	Warning messages from command line: <line>
RECOVERY	Signals that the execution of the command <line> results in a warning. In general there are other messages on the message stack which explain the source of the error. Correct the problem and reenter the command line.
SEVERITY	WA
Author	Walter F.J. Mueller

WARNING

MESSAGE	Warning condition: <syndrome>
RECOVERY	No recovery possible. Signals, that a general warning condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	WA
Author	Walter F.J. Mueller

GOODE

Prefix	XDE_
Object name	GOOMSGLIB(XDE)
File name	XDE.MSG
Dataset	RZ88.XDE.MSG
Identifier	V01.00A
Facility	109

ANLTABFULL

MESSAGE	the table <table> is full, <no> entries exceeded
RECOVERY	automatic table expansion should be implemented
SEVERITY	Error
Author	K.Winkelmann

CAMOUTRANGE

MESSAGE	CAMAC spectrum out of range. Offset/size= offset/size
RECOVERY	It was attempted to create a CAMAC spectrum with a location in the CAMAC MR2000 which is outside the valid range. MR2000 addresses range from 0 to 65536. Correct the offset parameter and spectrum size in the CRE SPEC command.
SEVERITY	ERROR
Author	H.G. Essel

CONTRARGS

MESSAGE	contradictory arguments were used: <args>
RECOVERY	check with the command description and invoke then correctly
SEVERITY	Error
Author	K.Winkelmann

CREERROR

MESSAGE	Error creating object.
RECOVERY	There are more messages specifying the reason of the error.
SEVERITY	ERROR
Author	H.G. Essel

DIMTOOBIG

MESSAGE	dimension specified (dim) is too big, up to <dimmax> is allowed.
RECOVERY	specify an allowed dimension
SEVERITY	Error
Author	K.Winkelmann

ERROR

MESSAGE	General warning message.
RECOVERY	Specified in message text.
SEVERITY	WARNING
Author	W. Spreng

ILLBINSIZE

MESSAGE	illegal binsize (<binsize>) was encountered

RECOVERY	binsize was zero,chose a value bigger than that
SEVERITY	Error
Author	K.Winkelmann

ILLNAME

MESSAGE	Illegal name encountered: 'name'
RECOVERY	Correct the name: watch position of colon and if name is complete.
SEVERITY	Error
Author	K.Winkelmann

INVALIDBITPATT

MESSAGE	invalid bit pattern encountered <charstring>
RECOVERY	check input, pattern is not of the form '10...'B or 10...
SEVERITY	Error
Author	K.Winkelmann

NOCREACOND

MESSAGE	Condition 'cond = spec' could not be created in the Data Base 'base'
RECOVERY	The previously shown error inhibited the creation of a condition. Try to avoid the previously shown error.
SEVERITY	ERROR
Author	M. Richter

NOCREAPOLY

MESSAGE	Polygon 'name' could not be created in the Data Base 'base'
RECOVERY	The previously shown error inhibited the creation of a polygon. Try to avoid the previously shown error.
SEVERITY	ERROR

Author	H.G.Essel

NOCREASPEC

MESSAGE	Spectrum 'name' could not be created in the Data Base 'base'
RECOVERY	The previously shown error inhibited the creation of a spectrum. Try to avoid the previously shown error.
SEVERITY	ERROR
Author	H.G.Essel

NODEL

MESSAGE	Data element <name> could not be deleted because of outstanding links or locks>
RECOVERY	The data element is either locked by an analysis process or display. Or it is still referenced in a dynamic list.
SEVERITY	WARNING
Author	H.G.Essel

NOTFOUND

MESSAGE	No <name of item> found
RECOVERY	requested items were not found
SEVERITY	Warning
Author	K.Winkelmann

NYI

MESSAGE	The feature <feature> has not yet been implemented.
RECOVERY	Bother the system programmers to get it implemented and try again.
SEVERITY	Warning
Author	K.Winkelmann

ODDNOLIMITS

MESSAGE	The limits given are '!AS', but an odd number of limits is not allowed.
RECOVERY	Specify an even no of limits(two for each dimension) and try again.
SEVERITY	Error
Author	K.Winkelmann

PTRNULL

MESSAGE	Passed pointers <variable names> are invalid (=NULL())
RECOVERY	check values of arguments in calling procedures
SEVERITY	Error
Author	K.Winkelmann

STRTOOLONG

MESSAGE	string <str> is <no> characters long, but the allowed maximum length is <maxno>.
RECOVERY	specify a shorter string
SEVERITY	Error
Author	K.Winkelmann

TOOMANY

MESSAGE	too many <ITEMS> have been specified string was truncated
RECOVERY	specify less items
SEVERITY	Warning
Author	K.Winkelmann

TOOMANYDIMS

MESSAGE	No of dimensions calculated is !AS, which is greater than the allowed maximum of 8.
RECOVERY	Specify less limits (though an even number) and try again.
SEVERITY	Error
Author	K.Winkelmann

U$ISCAN

MESSAGE	the token table for U$ISCAN while scanning <expr> is too small: <no>
RECOVERY	increase the dimensionality of the token table in the program calling U$ISCAN
SEVERITY	Error
Author	K.Winkelmann

WRSPECTYP

MESSAGE	wrong spectrum type <type> encountered, only N,H,I,L,R and D are allowed
RECOVERY	the wrong type is written into the spectrum header or given by the user
SEVERITY	Error
Author	K.Winkelmann

GOODISP

Prefix	XDISP_
Object name	GOOMSGLIB(XDISP)
File name	GOO$MSG:XDISP.MSG
Dataset	RZ88.XDISP.MSG
Identifier	V01.00A
Facility	108

ALLOCSTRUC

MESSAGE	Structure 'SD$PICTURE' not allocated
SEVERITY	WARNING
Author	W.Spreng

ALLOCSUCC

MESSAGE	Allocation successful! <TERM> is connected to physical address <_TXB5> and to workstationtype <TEK4014>
RECOVERY	-
SEVERITY	INFORMATION
Author	W. Spreng

AUTOSCAL

MESSAGE	Scaling parameter not specified. Autoscaling is performed.
RECOVERY	Interactive : Correct the input file. Program : Keyword ignored.

SEVERITY	INFORMATION
Author	W. Spreng

CURINPINT

MESSAGE	Cursor input interupted.
SEVERITY	WARNING
Author	W. Spreng

CURINPNONINT

MESSAGE	Cursor input non-interuptable. You have to specify <2> valid input values.
SEVERITY	WARNING
Author	W. Spreng

DEFDEVCRE

MESSAGE	Main device created. <TEK4014> is is used as default GOOSY display device
RECOVERY	-
SEVERITY	INFORMATION
Author	W. Spreng

DEVALLOC

MESSAGE	Device already allocated by an other user
RECOVERY	-
SEVERITY	WARNING
Author	W. Spreng

DEVALRALLOC

MESSAGE	Device already allocated by the same process

RECOVERY	-
SEVERITY	WARNING
Author	W. Spreng

DEVREQCAN

MESSAGE	Device request canceled
RECOVERY	-
SEVERITY	WARNING
Author	W. Spreng

ERROPENDEV

MESSAGE	Error while opening the specified device
RECOVERY	Interactive : Try allocation again Program : Cancel allocation
SEVERITY	WARNING
Author	W. Spreng

FATALERR

MESSAGE	-
SEVERITY	WARNING
Author	W. Spreng

FIRSTFRAME

MESSAGE	The keyword FRAME has to be specified after NOFRAMES
RECOVERY	Correct the input file.
SEVERITY	WARNING
Author	W. Spreng

FRAMENOTDEF

MESSAGE	Specified frame not defined
SEVERITY	WARNING
Author	W. Spreng

FRAMESPEC

MESSAGE	The keyword SPECTRUM is not specified after FRAME
RECOVERY	Correct the input file.
SEVERITY	WARNING
Author	W. Spreng

GKSRROR

MESSAGE	GKS-Error: '8' in routine 'GQTXF'
SEVERITY	WARNING
Author	W.Spreng

INCWKIDDEV

MESSAGE	Selected workstation identifier is inconsistant with the index in the device description table
RECOVERY	-
SEVERITY	WARNING
Author	W. Spreng

INITSUCCESS

MESSAGE	GOOSY-display process successfully initialised
RECOVERY	

SEVERITY	SUCCESS
Author	W.Spreng

INVDETYPE

MESSAGE	Dataelement <node::base:[dir]name> has an invalid type. <It is not a spectrum>.
SEVERITY	WARNING
Author	W. Spreng

INVDEVTYPE

MESSAGE	Specified device type: <UNKNOWN> is not supported in this GKS implementation
RECOVERY	Interactive : Allocate an other device Program : Cancel allocation
SEVERITY	WARNING
Author	W. Spreng

INVDISPMODE

MESSAGE	Invalid display mode: <PRIM> in frame <3>
RECOVERY	Enter correct display mode.
SEVERITY	WARNING
Author	W. Spreng

INVFRAME

MESSAGE	Invalid Frame number <200> specified or selected
SEVERITY	WARNING
RECOVERY	Specify valid frame
Author	W. Spreng

INVKEYWORD

MESSAGE	Invalid keyword: <XYRANGE>
RECOVERY	Interactive : Correct the input file. Program : Keyword ignored.
SEVERITY	INFORMATION
Author	W. Spreng

INVLOGNAME

MESSAGE	Invalid logical name: <DEVICE1>
RECOVERY	Interactive : Try allocation again Program : Cancel allocation
SEVERITY	WARNING
Author	W. Spreng

INVNUMSPEC

MESSAGE	Invalid number of spectra specified
RECOVERY	Correct the input file.
SEVERITY	WARNING
Author	W. Spreng

INVPIC

MESSAGE	Picture name: <PICTURE1> not defined.
RECOVERY	Interactive : Create new picture Program : Cancel picture request
SEVERITY	WARNING
Author	W. Spreng

INVSPENAME

MESSAGE	Invalide spectrum name: <SPEC1>
RECOVERY	Correct the input file.
SEVERITY	WARNING
Author	W. Spreng

INVVERSION

MESSAGE	Specified Dataelement <for spectrum in frame 1> has been modified. It will be ignored. <SERVITY: Specify new spectrum for this frame.>
SEVERITY	WARNING
Author	W. Spreng

INVWINDOW

MESSAGE	Invalid <temporary> window limits <,window not defined>
SEVERITY	INFORMATIONAL
RECOVERY	Define propper window limits
Author	W. Spreng

LIMDEV

MESSAGE	Maximum number of devices allocated
RECOVERY	Interactive : Deallocate other devices Program : Cancel allocation
SEVERITY	WARNING
Author	W. Spreng

MAXFRAME

MESSAGE	Number of allowed frames is limited on <100>

SEVERITY	INFORMATION
Author	W. Spreng

METAOUT

MESSAGE	Specified device is VMS-file Metafile output will be created
RECOVERY	-
SEVERITY	INFORMATION
Author	W. Spreng

MISARREL

MESSAGE	Missing name array element in picture-data-element
SEVERITY	WARNING
Author	W. Spreng

NOCALDEF

MESSAGE	No calibration defined
SEVERITY	WARNING
Author	W. Spreng

NODEVACTIVE

MESSAGE	No graphical device activated. Graphical output impossible, allocate a graphical device and try it again.
SEVERITY	WARNING
RECOVERY	Allocate a graphical GKS-device.
Author	W. Spreng

NODEVTAB

MESSAGE	Allocation of device-description table impossible. Main device tables does not exist.

RECOVERY	Interactive : Display must be initialized Program : Check display status. If other errors occur than display have to be initialized. If not,allocate device-table.
SEVERITY	WARNING
Author	W. Spreng

NOFRAMES

MESSAGE	You forgot to specify the number of frames
RECOVERY	Correct the input data.
SEVERITY	WARNING
Author	W. Spreng

NOINPDEVACT

MESSAGE	No input device active. Input impossible.
SEVERITY	WARNING
Author	W. Spreng

NOKEYWORD

MESSAGE	No keyword specified in line: 'XMODE'
SEVERITY	WARNING
Author	W. Spreng

NOOPGKS

MESSAGE	GKS is not opened
RECOVERY	Interactive: Display must be initialized Program : Check display status. If other errors occur than display have to be initialized. If not open GKS.

SEVERITY WARNING

Author W.Spreng

NOOVERLAYDEF

MESSAGE No overlayed spectra or scatterplots defined

SEVERITY WARNING

Author W. Spreng

NOPICDATA

MESSAGE Allocation of picture description table impossible. Picture data element does not exist.

RECOVERY
```
Interactive : Display must be initialized
  Program : Check display status. If other errors
                 occur than display have to be
                 initialized. If not allocate picture
                 data element.
```

SEVERITY WARNING

Author W. Spreng

NOPICHEAD

MESSAGE No picture header found

SEVERITY WARNING

Author W. Spreng

NOSPECTRUM

MESSAGE No spectrum specified <for frame 11>

RECOVERY Wrong command input => specify spectrum in command line
Spectrum not specified in picture => Modify frame.

SEVERITY WARNING

Author	W. Spreng

NOTALLFRA

MESSAGE	Not all defined frames are specified <10> frames are allocated, but you specified only <5> frames.
RECOVERY	Interactive : Specify the rest of frames Program : Command ignored
SEVERITY	WARNING
Author	W. Spreng

NOTBLENTRY

MESSAGE	No entry in the device description table found for the specified device
SEVERITY	WARNING
Author	W. Spreng

NTNOTDEF

MESSAGE	Normalisation transformation number <0> not defined
SEVERITY	WARNING
Author	W. Spreng

ONEDIMSPEC

MESSAGE	Stupid Command: <ZMODE> for one dimensional spectra. Command ignored.
RECOVERY	Interactive : Correct the input file. Program : Keyword ignored.
SEVERITY	INFORMATION
Author	W. Spreng

ONLYONEPARA

MESSAGE	Only one parameter specified.
SEVERITY	WARNING
Author	W. Spreng

PICEXISTS

MESSAGE	Picture 'LIVE' already exists
SEVERITY	INFORMATION
Author	W.Spreng

PICNAMENOTSPEC

MESSAGE	Picture name not specified
SEVERITY	WARNING
Author	W.Spreng

PROTPICNAME

MESSAGE	Protected picture name 'SPECTRUM'
SEVERITY	WARNING
Author	W.Spreng

READSAVE

MESSAGE	Screen image read from saveset <EE$ROOT:[USER.SAVESET]M.DAT>
SEVERITY	INFORMATION
Author	W. Spreng

REDUCED

MESSAGE	Reduced display action is active.

SEVERITY	WARNING
RECOVERY	Give DEFINE DISPLAY VERSION /normal command.
Author	W. Spreng

SCATTERFRAME

MESSAGE	Specified frame is a scatter-plot frame
SEVERITY	WARNING
Author	W. Spreng

SPECONEDIM

MESSAGE	Specified spectrum is only one dimensional
SEVERITY	WARNING
Author	W. Spreng

SUBSCRIPTERROR

MESSAGE	Subscripterror: array dimension to small
SEVERITY	WARNING
Author	W. Spreng

TOOMUCHFRASPEC

MESSAGE	You allocated '10' frames, but you specified more than '10'
RECOVERY	Structure SD$EDIT will be deleted
SEVERITY	WARNING
Author	W. Spreng

TWOFRAMES

MESSAGE	You selected two diffrent frames

SEVERITY	WARNING
Author	W. Spreng

WRGQUEUECRTBL

MESSAGE	Wrong queue cross-reference table for frame <2>
SEVERITY	WARNING
Author	W. Spreng

WRITSAVE

MESSAGE	Screen image written to saveset <EE$ROOT:[USER.SAVESET]M.DAT>
SEVERITY	INFORMATION
Author	W. Spreng

WRONGDIM

MESSAGE	Specified frame is a scatter-plot frame
SEVERITY	WARNING
Author	W. Spreng

GOODM

Prefix	XDM_
Object name	GOOMSGLIB(XDM)
File name	XDM.MSG
Dataset	RZ88.XDM.MSG
Identifier	V01.00A
Facility	102

AAILLARG

MESSAGE	Illegal arguments to allocate in AREA L_REQ: 'requested bytes', I_AREA_SIZE: 'area size'
RECOVERY	Error in calling routine.
SEVERITY	Fatal
Author	M. Richter

AANOSPACE

MESSAGE	No space to allocate data in an AREA
RECOVERY	AREA has no more space to allocate data in it.
SEVERITY	Fatal
Author	M. Richter

ADADIRNOTMAP

MESSAGE	AREA Directory of Data Base 'name' was not attached (mapped) successfully

RECOVERY	The Data Base might be corrupt.
SEVERITY	Error
Author	M. Richter

ADCDIRNOTMAP

MESSAGE	Master Directory of Data Base 'name' was not attached (mapped) successfully
RECOVERY	The Data Base might be corrupt.
SEVERITY	ERROR
Author	M. Richter

ADDBNOTATT

MESSAGE	The Data Base 'name' could not be attached
RECOVERY	The Data Base might be corrupt.
SEVERITY	ERROR
Author	M. Richter

AFILLARG

MESSAGE	Illegal arguments to free in AREA L_REQ: 'bytes', O_START_OFF: 'offset', I_AREA_SIZE: 'area size'
RECOVERY	Error in calling routine.
SEVERITY	Fatal
Author	M. Richter

AIILLARG

MESSAGE	Illegal arguments to initialize an AREA L_NBYTES_BIT: 'cluster', I_AREA_SIZE: 'area size'

RECOVERY	Error in calling routine.
SEVERITY	Fatal
Author	M. Richter

ALLDBMCNOTALL

MESSAGE	Mapping context structure for all Data Bases, SM$ALL_DBMC, is not allocated
RECOVERY	There was no M$ATDB done before calling the procedure
SEVERITY	Error
Author	M. Richter

ALREX

MESSAGE	<Module>: <object> already exists in <item>
RECOVERY	The procedure <module> tried to create the <object> in the library or Pdata base specified by <item> and found the object already exist.
SEVERITY	ERROR
Author	H.G.Essel

ARDIRNOTMAP

MESSAGE	The Area Directory of Data Base 'base' is not attached (mapped)
RECOVERY	The Data Base must be attached, before any other actions can be done.
SEVERITY	ERROR
Author	H.G.Essel

BADTYDSCR

MESSAGE	<Module>: The type descriptor validity pattern has a wrong value

RECOVERY	The type descriptor must be copied from the data base by M$TYGET to a local copy. The module got a pointer to something which is no type descriptor or it points direct into the Data Base.
SEVERITY	ERROR
Author	H.G.Essel

BMCILLARG

MESSAGE	Illegal arguments to clear bits in Bit Map L_REQ: 'req', L_POS: 'pos', BM_LENGTH: 'length'
RECOVERY	Correct call arguments
SEVERITY	Fatal
Author	M. Richter

BMNOSPACE

MESSAGE	No space to set bits in Bit Map L_REQ: 'req', BM_LENGTH: 'length'
RECOVERY	Correct call arguments
SEVERITY	Error
Author	M. Richter

BMNOTSET

MESSAGE	Bits to clear were not set in Bit Map L_REQ: 'req', L_POS: 'pos', BM_LENGTH: 'length'
RECOVERY	Correct call arguments
SEVERITY	Error
Author	M. Richter

BMSILLARG

MESSAGE	Illegal arguments to set bits in Bit Map L_REQ: 'req', BM_LENGTH: 'length'

RECOVERY	Correct call arguments
SEVERITY	Fatal
Author	M. Richter

CAAREADNOENT

MESSAGE	AREA Directory of Data Base 'name' has no more free entries
RECOVERY	The AREA Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CAAREADNOTMAP

MESSAGE	AREA Directory of Data Base 'name' is not attached (mapped).
RECOVERY	The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.
SEVERITY	Fatal
Author	M. Richter

CAAREANOTINI

MESSAGE	AREA header of AREA 'name' in Data Base 'base name' could not be initialized
RECOVERY	Error in procedures called by M$CRAR.
SEVERITY	Error
Author	M. Richter

CAAREATOOBIG

MESSAGE	Area 'area' of Data Base 'name' is too big. Too many clusters: 'clusters'

RECOVERY	It was attempted to create an area which is too big because the bits in the bitmap exceed 32K. Enlarge the clustersize, if the area belongs to a pool of your own (max. is 512 bytes per bit). If the area belongs to a directory, diminish the number of entries (sorry).
SEVERITY	Fatal
Author	M. Richter

CAARNAM2LONG

MESSAGE	AREA name 'name' is too long. Maximal number of characters is: 'n'
RECOVERY	Create an AREA with another name. The creation of an AREA should never be done explicitly by command but always indirectly, e.g. by creating a Data Element.
SEVERITY	Error
Author	M. Richter

CAARNAMEXISTS

MESSAGE	AREA name 'name' exists already in AREA Directory of Data Base 'base name'.
RECOVERY	Create an AREA with another name. The creation of an AREA should never be done explicitly by command but always indirectly, e.g. by creating a Data Element.
SEVERITY	Error
Author	M. Richter

CAARNAMNOTINS

MESSAGE	AREA name 'name' could not be inserted in AREA Directory of Data Base 'base name'. There might be not enough space in the AREA Directory.
RECOVERY	The AREA Directory must be created with a larger size. This should be done automatically.

SEVERITY	Error
Author	M. Richter

CADDBILLEGAL

MESSAGE	Data Base 'name' is illegal. The Home Block was not initialized correctly.
RECOVERY	Corrupt Data Base. Create a new one.
SEVERITY	Fatal
Author	M. Richter

CAILLCLUSIZ

MESSAGE	Illegal Area cluster size: 'size' (max. 'max') to create Area '!AS' for Data Base '!AS'
RECOVERY	Wrong cluster size defined in the argument list. Correct cluster size argument to a positive number with a maximum value of 512 bytes.
SEVERITY	Fatal
Author	M. Richter

CAILLPOOLIND

MESSAGE	Illegal Pool Directory index: 'pool index' of Data Base 'name', max. index: 'max. index'
RECOVERY	Data Base inconsistency or procedure error.
SEVERITY	Fatal
Author	M. Richter

CANODBSPACE

MESSAGE	No space in Data Base 'dbname' for new AREA 'arname' with 'n' pages

RECOVERY	The Data Base is filled. There is not enough contiguous space in the Data Base to create an AREA of the requested size. Create a new Data Base or request a smaller AREA.
SEVERITY	Error
Author	M. Richter

CANOPOOLDIR

MESSAGE	Pool Directory cannot be found in Data Base 'name'
RECOVERY	Corrupt Data Base. Create a new one.
SEVERITY	Fatal
Author	M. Richter

CAPOOLDNOTMAP

MESSAGE	Pool Directory of Data Base 'name' is not attached (mapped)
RECOVERY	The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.
SEVERITY	Fatal
Author	M. Richter

CDAREANOTCR

MESSAGE	Directory AREA 'name' could not be created for Data Base 'base name'.
RECOVERY	Any error in the M$CRAR procedure occured.
SEVERITY	Error
Author	M. Richter

CDAREANOTINI

MESSAGE	AREA header of Directory AREA 'name' could not be initialized in Data Base 'base name'

RECOVERY	Error in procedures called by M$CRAR.
SEVERITY	Error
Author	M. Richter

CDBATT

MESSAGE	Data Base 'name' is attached already. It could not be created again.
RECOVERY	Choose another name for your new Data Base
SEVERITY	Error
Author	M. Richter

CDDIRNAM2LONG

MESSAGE	Directory name 'name' is too long. Maximal number of characters is: 'max number'
RECOVERY	Create a Directory with another name. The creation of a Directory should never be done explicitly by command but always indirectly, e.g. by creating a Data Element.
SEVERITY	Error
Author	M. Richter

CDDIRNAMEXISTS

MESSAGE	Directory name 'name' exists already in Master Directory of Data Base 'base name'.
RECOVERY	Try another name.
SEVERITY	Error
Author	M. Richter

CDDIRNAMNOTINS

MESSAGE	Data Element Directory name 'name' could not be inserted in Master Directory. There might be not enough space in the Master Directory of Data Base 'base name'.
RECOVERY	The Master Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CDMASTDIRNOENT

MESSAGE	Master Directory of Data Base 'name' has no more free entries
RECOVERY	The Master Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CDMASTDIRNOTMAP

MESSAGE	Master Directory of Data Base 'name' is not attached (mapped)
RECOVERY	The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.
SEVERITY	Fatal
Author	M. Richter

CDNODBSPACE

MESSAGE	No space in Data Base 'name' for new Directory AREA 'dir. name' with 'n' pages

RECOVERY	The Data Base is filled. There is not enough contiguous space in the Data Base to create an AREA of the requested size for a new Directory. Create a new Data Base or request a smaller AREA.
SEVERITY	Error
Author	M. Richter

CDNODEDIR

MESSAGE	The Directory 'name' could not be created because the corresponding Data Element Directory of Data Base 'base name' could not be created.
RECOVERY	Error in calling procedures.
SEVERITY	Error
Author	M. Richter

CEAREADFUL

MESSAGE	AREA 'name' for data element 'name' cannot be created, because AREA directory of Data Base 'base' is full!
RECOVERY	A new area could not be created because the area directory is full. Reformat the data base with more entries for the area directory or enlarge the pools. If the pools are too small, many areas are created.
SEVERITY	Fatal
Author	M. Richter

CEAREADNOTMAP

MESSAGE	AREA Directory of Data Base 'name' is not attached (mapped)
RECOVERY	The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.
SEVERITY	Fatal
Author	M. Richter

CEDDBILLEGAL

MESSAGE	Data Base 'name' is illegal. The Home Block was not initialized correctly.
RECOVERY	Corrupt Data Base. Create a new one.
SEVERITY	Fatal
Author	M. Richter

CEDEDESCNOTINS

MESSAGE	Data Descriptor for Data Element 'name' could not be inserted in Data Element Directory 'dir name' of the Data Base 'base name'. There might be not enough space in the Directory.
RECOVERY	The Data Element Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CEDEDIFFNAMARR

MESSAGE	The requested Data Element name 'de' has inconsistent name array parameters compared to the existing in Data Element Directory 'dir' of Data Base 'db' and could not be replaced
RECOVERY	Try another name for the Data Element to replace an existing Data Element. Wrong argument for procedure M$CRDE.
SEVERITY	Error
Author	M. Richter

CEDEDNOENT

MESSAGE	Data Element Directory 'name' of Data Base 'b-name' has no more free entries

RECOVERY	The Data Element Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CEDENAM2LONG

MESSAGE	Data Element name 'name' is too long. Maximal number of characters is: 'max number'
RECOVERY	Create a Data Element with another name.
SEVERITY	Error
Author	M. Richter

CEDENAMARR

MESSAGE	Data Element name 'de' does not exists as a name array in Data Element Directory 'dir' of Data Base 'base' and could not be replaced
RECOVERY	Try another name for the Data Element to replace an existing Data Element. Wrong argument for procedure M$CRDE.
SEVERITY	Error
Author	M. Richter

CEDENAMARRNOTINS

MESSAGE	Descriptor extent for Data Element name array 'name' could not be inserted in Data Element Directory 'dir name' of Data Base 'base name'. There might be not enough space in the Directory.
RECOVERY	The Data Element Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CEDENAMEXISTS

MESSAGE	Data Element name 'name' exists already in Data Element Directory 'name' of Data Base 'base name' or is ambiguous
RECOVERY	Try another name.
SEVERITY	Error
Author	M. Richter

CEDENAMNOEXISTS

MESSAGE	Data Element name 'de' does not exist in Data Element Directory 'dir' of Data Base 'base' and could not be replaced
RECOVERY	Try another name for the Data Element to replace an existing Data Element. Wrong argument for procedure M$CRDE.
SEVERITY	Error
Author	M. Richter

CEDENAMNOTINS

MESSAGE	Data Element name 'name' could not be inserted in Data Element Directory 'dir name' of Data Base 'base name'. There might be not enough space in the Directory.
RECOVERY	The Data Element Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CEDENOFRAR

MESSAGE	The data Area of the existing Data Element name 'de' in Data Element Directory 'dir' of Data Base 'db' could not be freed

RECOVERY	The existing Data Element could not be deleted for the replacement by a new one. The Data Base might be corrupted.
SEVERITY	Error
Author	M. Richter

CEDENOFREXT

MESSAGE	The Directory extents of the existing Data Element name 'de' in Data Element Directory 'dir' of Data Base 'db' could not be freed
RECOVERY	The existing Data Element could not be deleted for the replacement by a new one. The Data Base might be corrupted.
SEVERITY	Error
Author	M. Richter

CEDENOMPAR

MESSAGE	The data Area of the existing Data Element name 'de' in Data Element Directory 'dir' of Data Base 'db' could not be mapped
RECOVERY	The existing Data Element could not be deleted for the replacement by a new one. The Data Base might be corrupted.
SEVERITY	Error
Author	M. Richter

CEDENOMPTYAR

MESSAGE	The Type Descriptor Area for the existing Data Element name 'de' in Data Element Directory 'dir' of Data Base 'db' could not be mapped
RECOVERY	The existing Data Element could not be deleted for the replacement by a new one. The Data Base might be corrupted.
SEVERITY	Error
Author	M. Richter

CEDENONAM

MESSAGE The Data Element and the queue header was defined
with a null string name
for the Data Element Directory 'dir name' of Data
Base 'base name'.

RECOVERY A Data Element is queued only if a queue header was defined, otherwise it is not queued. In the last the Data Element must have its own name.

SEVERITY Error

Author M. Richter

CEDENONAMARR

MESSAGE Data Element name 'de' does not exist as a name array
in Data Element Directory 'dir'
of Data Base 'base' and could not be replaced

RECOVERY Try another name for the Data Element to replace an existing Data Element. Wrong argument for procedure M$CRDE.

SEVERITY Error

Author M. Richter

CEDENOQMEM

MESSAGE Data Element name 'de' is not a queue member
in Data Element Directory 'dir'
of Data Base 'base' and could not be replaced

RECOVERY Try another name for the Data Element to replace an existing Data Element. Wrong argument for procedure M$CRDE.

SEVERITY Error

Author M. Richter

CEDIRNOTFOUND

MESSAGE Directory 'name' not found in Master Directory
of Data Base 'base name'

RECOVERY Data Element could not be placed in the requested Data Element Directory. Create a new Data Element Directory with the requested name.

SEVERITY Error

Author M. Richter

CEILLCLUST

MESSAGE Data Element 'name' to be created in the Data Element Directory 'dir name' of Data Base
'base name'
has the illegal cluster size of 'nbytes_bit'.

RECOVERY Data Elements must have a positive cluster size.

SEVERITY Error

Author M. Richter

CEILLNAMARRDIM

MESSAGE The argument 'p_name_array_refer' to create the Data Element
name array 'array' in Data Element Directory 'dir'
of Data Base 'db'
defines illegal dimension limits.

RECOVERY The argument 'p_name_array_refer' of the procedure call M$CRDE must be corrected.

SEVERITY Error

Author M. Richter

CEMASTDIRNOTMAP

MESSAGE Master Directory of Data Base 'name' is not attached (mapped)

RECOVERY The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.

SEVERITY Fatal

Author M. Richter

CEMASTDIRNOTMAP

MESSAGE	Main Directories and Home Block of Data Base 'base' are not attached (mapped) with write access
RECOVERY	The Data Base was not attached with write access. M$ATDB was called with read access only.
SEVERITY	Error
Author	M. Richter

CENOALLAREA

MESSAGE	Data Element 'name' could not be allocated in the newly created AREA 'area name' of the Data Base 'base name'.
RECOVERY	
SEVERITY	Error
Author	M. Richter

CENODEAREA

MESSAGE	No space for area 'name' for Data Element 'element-name' in Data Base 'base name'.
RECOVERY	The Data Base is filled. There is not enough contiguous space in the Data Base to create an AREA of the requested size for a new Data Element. Create a new Data Base or request a smaller Data Element. An empty data element name or asterisk shows that a queued data element was to be created.
SEVERITY	Error
Author	M. Richter

CENODEAREANAME

MESSAGE	AREA for Data Element 'name' could not be created

	in the Data Base 'base name'. All AREAs up to 'AREA name' exists already in Pool 'pool name'.
RECOVERY	Use another Pool for the Data Element.
SEVERITY	Error
Author	M. Richter

CENOLENGTH

MESSAGE	Data Element 'name' to be created in the Data Element Directory 'dir name' of Data Base 'base name' has the illegal length of 'length'.
RECOVERY	Only Data Elements of the Type 'Y' may have zero length. No Data Element may have a negative length.
SEVERITY	Error
Author	M. Richter

CENOREFER

MESSAGE	Data Element 'name' to be created in the Data Element Directory 'dir name' of Data Base 'base name' has wrong number of refer values: 'refer'.
RECOVERY	The data type declaration contained refer members. The number of refer values specified in CRE ELEM did not match the number of these members.
SEVERITY	Error
Author	M. Richter

CEPOOLDNOTMAP

MESSAGE	Pool Directory of Data Base 'name' is not attached (mapped)
RECOVERY	The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.

SEVERITY	Fatal
Author	M. Richter

CEPOOLNOTFOUND

MESSAGE	Pool 'name' not found in Pool Directory of Data Base 'base name'
RECOVERY	Data Element could not be placed in the requested Pool. Create a new Pool with the requested name.
SEVERITY	Error
Author	M. Richter

CEQNOTFOUND

MESSAGE	Data Element 'name' (queue header) not found in the Data Element Directory 'dir name' of Data Base 'base name'.
RECOVERY	The requested queue header does not exists. Change name or Data Element Directory.
SEVERITY	Error
Author	M. Richter

CEQNOTHEAD

MESSAGE	Data Element 'name' is not a queue header in the Data Element Directory 'dir name' of Data Base 'base name'.
RECOVERY	The requested queue header does not exists. Change name or Data Element Directory.
SEVERITY	Error
Author	M. Richter

CEQTYPAMB

MESSAGE The data type 'type' requested for the new Data Element 'de' in Data Element Directory 'dir' of Data Base 'base' exists already in the queue 'qname' (type ambiguity)

RECOVERY The argument 'cv_type_name' of the procedure call M$CRDE must be corrected. The creation of a new Data Element which should be member of a Data Element queue was requested. The Data Type of the new Data Element exists already in the queue. Such type ambiguities are not allowed in a Data Element queue. You have to choose another existing Data Type or you have to create a new one.

SEVERITY Error

Author M. Richter

CEQTYPNOREP

MESSAGE The Data Element 'element' to replace in
Data Element Directory 'dir' of Data Base 'base'
is a queue member or head. The Data Type 'type'
is not the same as the existing. Replacement
impossible.

RECOVERY The type argument of the procedure call M$CRDE must be corrected. If a single Data Element or a Data Element name array should be replaced, the type of the new one can only change if the Data Element or any of the name array is not a queue member or a queue head. If so, the type must be the same as for the current.

SEVERITY Error

Author M. Richter

CETYPEDNOTMAP

MESSAGE Type Directory of Data Base 'name' is not attached (mapped)

RECOVERY The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.

SEVERITY	Fatal
Author	M. Richter

CETYPNOTFOUND

MESSAGE	Data Element Type 'name' not found in the Type Directory of Data Base 'base name' for Data Element 'e-name'.
RECOVERY	Request a legal Data Element Type.
SEVERITY	Error
Author	M. Richter

CMDDBILLEGAL

MESSAGE	Data Base 'name' is illegal. The Home Block was not initialized correctly.
RECOVERY	Corrupt Data Base. Create a new one.
SEVERITY	Fatal
Author	M. Richter

COPYNOTALLOWED

MESSAGE	<Module>: Copy from source Datatype <datatype> to target Datatype <datatype> not allowed.
RECOVERY	Correct source or target Data Element member.
SEVERITY	Error
Author	T. KROLL

CPDDBILLEGAL

MESSAGE	Data Base 'name' is illegal. The Home Block was not initialized correctly.
RECOVERY	Corrupt Data Base. Create a new one.

SEVERITY Fatal

Author M. Richter

CPPONAM2LONG

MESSAGE Pool name 'name' is too long.
Maximal number of characters is: 'max nunmber'

RECOVERY Create a Pool with another name.

SEVERITY Error

Author M. Richter

CPPONAMEXISTS

MESSAGE Pool name 'name' exists already in Pool Directory
of Data Base 'base name'

RECOVERY Try another name.

SEVERITY Error

Author M. Richter

CPPONAMNOTINS

MESSAGE Pool name 'name' could not be inserted in Pool Directory.
There might be not enough space in the Pool
Directory
of the Data Base 'base name'.

RECOVERY The Pool Directory must be created with a larger size. This should be done automatically.

SEVERITY Error

Author M. Richter

CPPOOLDNOENT

MESSAGE Pool Directory of Data Base 'name' has no more free entries

RECOVERY	The Pool Directory must be created with a larger size. This should be done automatically.
SEVERITY	Error
Author	M. Richter

CPPOOLDNOTMAP

MESSAGE	Pool Directory of Data Base 'name' is not attached (mapped)
RECOVERY	The Data Base was not attached normally. M$ATDB might not be call before. Check procedures.
SEVERITY	Fatal
Author	M. Richter

CRLIDENOTUSE

MESSAGE	Illegal Data Element index 'index' to create Data Element links in Data Base 'base'. A Data Element with this index is not in use.
RECOVERY	Calling argument of M$CRLI l_from_de_index or l_to_de_index must be corrected. Such a Data Element was not created in the Data Base. Therefore this entry of the Data Element Directory is not in use.
SEVERITY	Error
Author	M. Richter

CRLIDIRNOTUSE

MESSAGE	Illegal Data Element Directory index 'index' to create Data Element links in Data Base 'base'. A Directory with this index is not in use.
RECOVERY	Calling argument of M$CRLI l_from_dir_index or l_to_dir_index must be corrected. Such a Directory was not created in the Data Base. Therefore this entry of the Master Directory is not in use.
SEVERITY	Error
Author	M. Richter

CRLIILLDEINDEX

MESSAGE	Illegal Data Element index 'index' to create Data Element links in Data Base 'base'. An index must be positive with a maximum of 'max'.
RECOVERY	Calling argument of M$CRLI l_from_de_index or l_to_de_index must be corrected.
SEVERITY	Error
Author	M. Richter

CRLIILLDIRINDEX

MESSAGE	Illegal Data Element Directory index 'index' to create Data Element links in Data Base 'base'. An index must be positive with a maximum of 'max'.
RECOVERY	Calling argument of M$CRLI l_from_dir_index or l_to_dir_index must be corrected.
SEVERITY	Error
Author	M. Richter

DBCOMPRESS

MESSAGE	Error during data base compress: <description>
RECOVERY	An error occured during the compression of a GOOSY data base. The data base must not be mounted.
SEVERITY	ERROR
Author	H.G.Essel

DBCOPY

MESSAGE	Error during data base copy: <description>
RECOVERY	An error occured during the copy of a GOOSY data base. The data base must not be mounted.

SEVERITY	ERROR
Author	H.G.Essel

DBDECOMPRESS

MESSAGE	Error during data base decompress: <description>
RECOVERY	An error occured during the decompression of a GOOSY data base. The data base must not be mounted.
SEVERITY	ERROR
Author	H.G.Essel

DBFEXIS

MESSAGE	Data Base Section file: 'file' exists already
RECOVERY	You may have required a wrong Section File. If so, create another Data Base with the right Section file
SEVERITY	Information
Author	M. Richter

DBGSEXIS

MESSAGE	Data Base 'db' exists already and is defined as a (System) Global Section
RECOVERY	None
SEVERITY	Information
Author	M. Richter

DBILLNAME

MESSAGE	Data Base name 'name' is a NULL string or too long. Maximal number of characters is: 'n'
RECOVERY	Create or attach a Data Base with another name.

SEVERITY	Error
Author	M. Richter

DBMCNOTALL

MESSAGE	No Data Base attached at all
RECOVERY	Before any calls to Data Base routines, M$ATDB must be called to allocate the local mapping context.
SEVERITY	ERROR
Author	H.G.Essel

DBNEWILLVERS

MESSAGE	Illegal version number: 'n'. The Data Base 'name' cannot be initialized.
RECOVERY	The version of the Data Base was not supported. Error in calling procedure.
SEVERITY	Fatal
Author	M. Richter

DBNEWSHORT

MESSAGE	Length of Data Base 'name' mapped is ONLY 'm' pages instead of 'n'
RECOVERY	The Data Base could not be created with the desired size. The reason might be the size of the Section File (disk space) or the virtual address space limit of the requesting process.
SEVERITY	Error
Author	M. Richter

DBNOALL

MESSAGE	Pages in the Data Base 'name' could not be allocated and mapped

RECOVERY	The Data Base might be corrupt.
SEVERITY	Error
Author	M. Richter

DBNOFREE

MESSAGE	Pages in the Data Base 'name' could not be freed and unmapped
RECOVERY	The Data Base might be corrupt.
SEVERITY	Error
Author	M. Richter

DBNOTATT

MESSAGE	Data Base 'name' is not yet attached to the process
RECOVERY	There was no M$ATDB done before calling the procedure
SEVERITY	Error
Author	M. Richter

DBOLDILLVERS

MESSAGE	The Data Base 'name' has the wrong version number: 'n'. Data Base is although reopened. To ignore it, create a new one.
RECOVERY	The specified file might not be a Section File of a Data Management Data Base. Create a new Data Base.
SEVERITY	Warning
Author	M. Richter

DBOLDSHORT

MESSAGE	Length of old Data Base 'name' mapped is ONLY 'm' pages instead of 'n'

RECOVERY	You may have required a wrong Section File. If so, create another Data Base with the right Section file
SEVERITY	Warning
Author	M. Richter

DBREOP

MESSAGE	The Data Base 'db' was not initialized but reopened
RECOVERY	The Data Base Section File exists already and was reopened. You may create a new Data Base with a new Section File.
SEVERITY	Warning
Author	M. Richter

DBTOOSHORT

MESSAGE	Length of requested Data Base 'name' is too short, only 'm' pages instead of 'm' for Home Block
RECOVERY	There are not enough pages in the (System) Global Section to be mapped. Depending on the primary try to create a new, larger Data Base.
SEVERITY	Error
Author	M. Richter

DCLDERR

MESSAGE	<Module>: Error building type descriptor: <name>
RECOVERY	There is something wrong with the calling of M$TYMEM Check argument lists.
SEVERITY	ERROR
Author	H.Essel

DEDELLINK

MESSAGE	The Data Element 'element' of the Data Element Directory 'dir' in the Data Base 'base' could not be deleted because it has outstanding links to other Data Elements.
RECOVERY	The Data Element has still links to other Data Elements in the Data Base. You must delete all incoming and outcoming links to a Data Element before you can delete the Data Element.
SEVERITY	Error
Author	M. Richter

DEDELMEM

MESSAGE	The Data Element 'element' of the Data Element Directory 'dir' in the Data Base 'base' could not be deleted because it is a member of a name array.
RECOVERY	The Data Element is a member of a name array in the Directory. You can only delete the whole array, but not single members of it.
SEVERITY	Error
Author	M. Richter

DEDELPROT

MESSAGE	The Data Element 'element' of the Data Element Directory 'dir' in the Data Base 'base' could not be deleted because it is protected.
RECOVERY	The Data Element is marked in the Directory with a deletion protection. To override this protection you must specify /UNPROTECT together with the delete command or you must set the B_SM$DLDE_PROT_FLAG in the call of the M$DLDE procedure. !!!! ATTENTION !!!! Be very careful using the protection override! You may corrupt the Data Base!

SEVERITY Error

Author M. Richter

DEDELQHEAD

MESSAGE The Data Element 'element'
of the Data Element Directory 'dir'
in the Data Base 'base' could not be deleted
because it is the head of a Data Element queue.

RECOVERY The Data Element is still the head of a Data Element Queue in the Directory. You must delete all other queued Data Elements of this queue before you can delete the Data Element.

SEVERITY Error

Author M. Richter

DEINDEXNOTUSE

MESSAGE A Data Element index 'element'
of the Data Element Directory 'dir'
in the Data Base 'base' is not in use.

RECOVERY An argument l_de_index was incorrectly given. Such a Data Element was not created or was deleted in the Data Base. Therefore this entry of the Data Element Directory is not in use.

SEVERITY Error

Author M. Richter

DENAMEXISTS

MESSAGE Data Element name 'de' does not exist as a name array
in Data Element Directory 'dir'
of Data Base 'base'.

RECOVERY Try another name for the Data Element to manipulate an existing Data Element. Wrong procedure argument.

SEVERITY Error

Author M. Richter

DENODATA

MESSAGE The Data Element with the index 'index'
in the Data Base 'base' has no data

RECOVERY A Data Element defined without data was located in a data base. The returned data pointer is set to NULL().

SEVERITY Error

Author M. Richter

DENONAMEARR

MESSAGE <Module> Target Data Element <data element> does not exists as an name array.

RECOVERY Correct input for target Data Element or create them.

SEVERITY Error

Author T. KROLL

DIMNOTEQUAL

MESSAGE <Module>: The destination Data Element <source data element> has another dimensionality the defined in source Data Element <destination data element>.

RECOVERY The specified source DE and destination DE name array dimensionality are not equal. Specify a another Data Element or use a another command to copy this Data Element.

SEVERITY Error

Author T. KROLL

DIRNOTMAP

MESSAGE <module>: Directory '<direct>' of data base 'base' is not attached (mapped)

RECOVERY	The directory must be attached, before any other actions can be done.
SEVERITY	ERROR
Author	H.G.Essel

DIRNOTWMAP

MESSAGE	The Data Element Directory 'dir' of Data Base 'base' is not attached (mapped) with write access
RECOVERY	The Directory was not attached with write access. M$ATDI was called with read access only.
SEVERITY	Error
Author	M. Richter

DLLIDENOTUSE

MESSAGE	Illegal Data Element index 'index' to delete Data Element links in Data Base 'base'. A Data Element with this index is not in use.
RECOVERY	Calling argument of M$DLLI l_from_de_index or l_to_de_index must be corrected. Such a Data Element was not created in the Data Base. Therefore this entry of the Data Element Directory is not in use.
SEVERITY	Error
Author	M. Richter, H.G.Essel

DLLIDIRNOTUSE

MESSAGE	Illegal Data Element Directory index 'index' to delete Data Element links in Data Base 'base'. A Directory with this index is not in use.
RECOVERY	Calling argument of M$DLLI l_from_dir_index or l_to_dir_index must be corrected. Such a Directory was not created in the Data Base. Therefore this entry of the Master Directory is not in use.
SEVERITY	Error
Author	M. Richter, H.G.Essel

DLLIILLDEINDEX

MESSAGE	Illegal Data Element index 'index' to delete Data Element links in Data Base 'base'. An index must be positive with a maximum of 'max'.
RECOVERY	Calling argument of M$DLLI l_from_de_index or l_to_de_index must be corrected.
SEVERITY	Error
Author	M. Richter, H.G.Essel

DLLIILLDIRINDEX

MESSAGE	Illegal Data Element Directory index 'index' to delete Data Element links in Data Base 'base'. An index must be positive with a maximum of 'max'.
RECOVERY	Calling argument of M$DLLI l_from_dir_index or l_to_dir_index must be corrected.
SEVERITY	Error
Author	M. Richter, H.G.Essel

DYNBREAK

MESSAGE	Dynamic list <name>: execution terminated. Reason: <specification of reason>
RECOVERY	Dynamic list execution may be terminated by two events: 1. A master condition resulted in FALSE 2. A master condition procedure failed.
SEVERITY	WARNING
Author	H.G.Essel

DYNEXEERR

MESSAGE	Error during execution of dynamic list: <dynamic list>

RECOVERY	Reinitialize dynamic list by updating it.
SEVERITY	WARNING
Author	H.G.Essel

DYNINIERR

MESSAGE Error during initialization of dynamic list:
<dynamic list>
<specification>

RECOVERY Check the specified dynamic list entry as created in the data base. Possible reasons for this error:

1. Any of the referenced data elements could not be found. This should be expressed by another message.
2. The number of objects does not match the dimensionality of the entry.
3. The data type of some parameter does not match the required type.

SEVERITY WARNING

Author H.G.Essel

DYNLFULL

MESSAGE	Module: Dynamic list <list name> full
RECOVERY	There are no more entries for sublists in the dynamic list. Create a new larger list.
SEVERITY	E
Author	H.G.Essel

DYNLISTAT

MESSAGE	There are dynamic lists active (attached).
RECOVERY	DETACH dynamic lists and try again the command returning the error.
SEVERITY	WARNING

Author H.G.Essel

DYNODEL

MESSAGE Dynamic list could not be deleted, because it is still locked.

RECOVERY The dynamic list is still in use by an analysis program. As long as an analysis program is attached to a dynamic list, it can not be deleted. The list entries, however, are deleted. If by error no more programs are attached to the list, but it is still locked, delete the links between the dynamic list and its buffer. Then delete the list.

SEVERITY WARNING

Author H.G.Essel@

DYNSYNERR

MESSAGE Syntax error in dynamic list specification: <list specification> <erraneous part of list specification>

RECOVERY Enter correct list specification:
  item,qual=list,qual=list,...
  list is a DE-specification or a list of DE-specs
enclosed in parenthesis',

SEVERITY W

Author H.G.Essel

ERRDIR

MESSAGE <Modul>: Error in directory structures.

RECOVERY There are more entries in the directory than specified in the header. Directory may be corrupt. Inform the GOOSY group.

SEVERITY ERROR

Author H. G. Essel

ERRNAM

MESSAGE <module>: Error in specified name '<name>'

RECOVERY	The name is too long or contains invalid characters
SEVERITY	ERROR
Author	H.Essel

IDNOARRAYHEAD

MESSAGE	The Data Element ID 'id' is not the head of a name array.
RECOVERY	The Data Element ID 'id' is not the head of a name array.
SEVERITY	Error
Author	Th. Kroll

ILLAREAINDEX

MESSAGE	Illegal Area Directory index: 'area index' of Data Base 'name', max. index: 'max. index'
RECOVERY	Data Base inconsistency or procedure argument error.
SEVERITY	Error
Author	M. Richter

ILLDBINDEX

MESSAGE	Illegal index to Data Base mapping context array : 'index'
RECOVERY	Correct call arguments
SEVERITY	Error
Author	M. Richter

ILLDEINDEX

MESSAGE	Illegal Data Element index 'element' in Directory 'dir' of Data Base 'base'. An index must be positive with a maximum of 'max'.
RECOVERY	A calling argument 'l_de_index' must be corrected.

SEVERITY	Error
Author	M. Richter

ILLDIRINDEX

MESSAGE	Illegal Data Element Directory index 'index' for Data Base 'base'. An index must be positive with a maximum of 'max'.
RECOVERY	Calling argument of l_dir_index must be corrected.
SEVERITY	Error
Author	M. Richter

ILLDYNINDEX

MESSAGE	<module>: Local dynamic list index out of range : #
RECOVERY	A dynamic list must be attached by M$ATDL, wich returnes the local dynamic list index.
SEVERITY	ERROR
Author	H.G.Essel

ILLSPECNARRMEM

MESSAGE	<Module>: The specified number of source Name Array <number of source name array member> are not equal to the number of destination Name Array <number of destination name array member> member.
RECOVERY	The specified number of source Name Array are not equal to the number of destination Name Array member. Correct the number (limit) of name array member for the source or destination Name array member.
SEVERITY	Error
Author	T. KROLL

INCONS

MESSAGE	<Module>: Inconsistent <object>

RECOVERY	Check the calls for <module> and the actual parameters. Some of them are specified in an invalid combination.
SEVERITY	ERROR
Author	H.G.Essel

INPUTTOBIG

MESSAGE	<Module>: The Input 'value' contains to many 'character' or 'bits'.
RECOVERY	Specify for input value correct character length.
SEVERITY	Error
Author	T. Kroll

INVCOPYOPER

MESSAGE	<Modul>: Invalid copy operation.
RECOVERY	Copying Data Elements to same Data Base and same directory are not allowed.
SEVERITY	ERROR
Author	Th. Kroll

INVDBID

MESSAGE	<Module>: The Data Base ID is out of range: <id>
RECOVERY	Check the call of <module>. Check for correct value of the Data Base ID. The ID is returned by M$ATDB or M$LOCDB and must not changed.
SEVERITY	ERROR
Author	H.E.Essel

INVDEID

MESSAGE	The Data Element entry index 'id' is not valid in the Directory 'dir' of Data Base 'base'.

RECOVERY	Check the value of the Data Element index as passed to the module. Note, that indeces must not be changed in programs.
SEVERITY	ERROR
Author	M. Richter

INVDELEMID

MESSAGE	<Module>: The Data Element entry ID <id> is not valid in Data Base <base>
RECOVERY	Check the value of the ID as passed to <module>. Note, that ID's must not be changed in programs.
SEVERITY	ERROR
Author	H.G.Essel

INVDIRID

MESSAGE	<Module>: The directory index <id> is not valid in Data Base <base>
RECOVERY	Check the value of the directory index as passed to module. Note, that indeces must not be changed in programs.
SEVERITY	ERROR
Author	H.G.Essel

INVPOOLID

MESSAGE	<Module>: The pool <id> is not valid in Data Base <base>
RECOVERY	Check the value of the pool index as passed to module. Note, that indeces must not be changed in programs.
SEVERITY	ERROR
Author	H.G.Essel

INVPTR

MESSAGE	<Module>: Invalid pointer to <object>

RECOVERY	An argument pointer variable has the value NULL. Check statements calling <module>. Check validity of pointers passed to <module>.
SEVERITY	ERROR
Author	H.Essel

INVSUBLIST

MESSAGE	Module: Dynamic sublist type '<sublist>' not supported
RECOVERY	Check the name of the sublist. Supported sublist types must be implemented.
SEVERITY	E
Author	H.G.Essel

INVTYPE

MESSAGE	A Data Type 'type name' could not be found in the Data Base 'base name'
RECOVERY	Check the type name as passed to module.
SEVERITY	ERROR
Author	M. Richter

INVTYPEID

MESSAGE	<Module>: The type entry ID <id> is not valid in Data Base <base>
RECOVERY	Check the value of the type ID as passed to module. Note, that ID's must not be changed in programs.
SEVERITY	ERROR
Author	H.G.Essel

INVTYPESPEC

MESSAGE	<Modul>: The Data Element 'DE' is a 'Type'.

RECOVERY	Check that the Data element has an correct type for the wanted operation.
SEVERITY	ERROR
Author	T. Kroll

INVWILDSPEC

MESSAGE	<Modul>: Wildcards for Spectrum not correct specified.
RECOVERY	Correct input for source or target Spectrum.
SEVERITY	ERROR
Author	Th. Kroll

LENOTFOUND

MESSAGE	Data Element 'name' not found in the Data Element Directory 'dir name' of Data Base 'base name'.
RECOVERY	A located Data Element does not exists. The name or the Data Element Directory might be wrong.
SEVERITY	Error
Author	M. Richter

LINKALREX

MESSAGE	<module>: Data Element link already exist: <link specification>.
RECOVERY	The specified link is not created. If You want to create multiple links, specify DMLI_MULTIPLE in M$CRLI.
SEVERITY	WARNING
Author	H.G.Essel

LINKNOTF

MESSAGE	<module>: Link not found: <Link specification>

RECOVERY	A link between the two data elements does not exist. This may be an error or not, depending on history.
SEVERITY	WARNING
Author	H.G.Essel

LOCKERR

MESSAGE	An error occured for the following Data Base lock: 'lock name'
RECOVERY	Another process might block this lock. Show locks.
SEVERITY	Error
Author	M. Richter

LOWER_GR_UPPERLIMIT

MESSAGE	<Module>: The upper limit 'value' must be greater then the lower limit 'value' for member 'member in Type Descriptor'.
RECOVERY	Specify for Input Member correct array limits.
SEVERITY	Error
Author	T. Kroll

LQEDIRNOTFOUND

MESSAGE	For Data Base 'base name' a Directory with the index '#' could not be found in Master Directory
RECOVERY	A Master Directory index for a Data Element Directory was invalid (less 0 or the large) or no Data Element Directory is defined under this entry.
SEVERITY	Error
Author	M. Richter

LQEHEADNOTFOUND

MESSAGE	For Data Base 'base name' a Data Element with the index '#' could not be found in the Data Element Directory with the index '#'

RECOVERY	A Data Element Directory index for a Data Element was invalid (less 0 or the large) or no Data Element is defined under this entry. The head of a Data Element queue could not be found.
SEVERITY	Error
Author	M. Richter

LQENOHEAD

MESSAGE	For Data Base 'base name' a Data Element with the index '#' is not a head of a queue in the Data Element Directory with the index '#'
RECOVERY	A Data Element found in a Data Element Directory is not the head of a Data Element Queue.
SEVERITY	Error
Author	M. Richter

LQENOQDE

MESSAGE	For Data Base 'base name' a queued Data Element with the Data Type 'type name' could not be found in the Directory with the index '#'
RECOVERY	A queued Data Element with a specific Data Type could not be found in a Data Element Directory.
SEVERITY	Error
Author	M. Richter

LTYPNOTFOUND

MESSAGE	Data Element Type 'name' not found in the Type Directory of Data Base 'base name'.
RECOVERY	Request a legal Data Element Type.
SEVERITY	Error
Author	M. Richter

MAARNOTUSE

MESSAGE	Illegal Area index 'index' to access an Area of Data Base 'base'. An Area with this index is not in use.
RECOVERY	Calling argument of M$MPAR or M$DLAR 'l_area_index' must be corrected. Such an Area was not created in the Data Base. Therefore this entry of the Area Directory is not in use.
SEVERITY	Error
Author	M. Richter

MAILLARINDEX

MESSAGE	Illegal Area index 'index' to map an Area of Data Base 'base'. An Area index must be positive with a maximum of 'max'.
RECOVERY	Calling argument of M$MPAR l_area_index must be corrected.
SEVERITY	Error
Author	M. Richter

MASTDIRNOTMAP

MESSAGE	The Master Directory of Data Base 'b' is not attached (mapped)
RECOVERY	The Data Base must be attached, before any other actions can be done.
SEVERITY	ERROR
Author	H.G.Essel

MDGSEXISTS

MESSAGE	The Data Base 'name' exists already as a (System) Global Section
RECOVERY	None

SEVERITY	Warning
Author	M. Richter

MDGSFILEERROR

MESSAGE	The Section file 'file name' for the Data Base 'name' to be mounted does not exist. Perform the DCL commands SHOW DEFAULT and DIR.
RECOVERY	The Section file does not exist under the directory or any directory or device error occured. Either use an existing Section file name or check the device and directory specifications or the default settings for the device and directory (perform the DCL commands: SHOW DEFAULT and/or DIR).
SEVERITY	Fatal
Author	M. Richter

MDGSWRONGFILE

MESSAGE	The Data Base 'name' exists already and is defined as a (System) Global Section (is already mounted), but with the Section file 'current file name' instead of the requested file 'requested file name'.
RECOVERY	The current Data Base is using a different Section file, i.e. Data Base file, then the new mount request. Either use the same Section (Data Base) file or use a different Data Base name (Section name). Otherwise the current Data Base would be corrupted.
SEVERITY	Fatal
Author	M. Richter

MDILLVERS

MESSAGE	The Data Base 'name' has the wrong version number: 'n'. The Data Base is although reopened. To ignore it, mount another one.
RECOVERY	Corrupt Data Base. Create a new one.

SEVERITY	Warning
Author	M. Richter

MDINODIR

MESSAGE	No Directory was found to map in the Data Base 'db'
RECOVERY	The Data Base is corrupt. Create a new one.
SEVERITY	Fatal
Author	M. Richter

MDISOMEDIR

MESSAGE	Only some Directories were found to map in the Data Base 'name'
RECOVERY	The Data Base is corrupt. Create a new one.
SEVERITY	Fatal
Author	M. Richter

MDLESSPAG

MESSAGE	Only 'np' pages instead of 'np' requested pages could be mapped for Data Base 'db_name'
RECOVERY	There are not enough pages in the (System) Global Section to be mapped. Depending on the primary try to create a new, larger Data Base.
SEVERITY	Warning
Author	M. Richter

MISMATCH

MESSAGE	!AS: '!AS' does not match '!AS'
RECOVERY	
SEVERITY	Error
Author	H.G.Essel

NAMNOTF

MESSAGE	<Module>: Name '<name>' not found in <item>
RECOVERY	Try to find out why a name is searched which does not exist. If <name> is a directory, pool, or type name, then the data base used is not built correctly. Note that pools and directories must be created before use.
SEVERITY	ERROR
Author	H.Essel

NARIDNOARRAYHEAD

MESSAGE	The Data Element 'name' is not the head of a name array.
RECOVERY	The argument 'p_sm$enam' of the procedure call M$NARID must be corrected. The name CV_SM$ENAM defined in the name array structure SM$ENAM is not name array head.
SEVERITY	Error
Author	M. Richter

NARIDNOEXT

MESSAGE	The Data Element 'name' has no Directory entry extents. Therefor it can't be the head of a name array.
RECOVERY	The argument 'p_sm$enam' of the procedure call M$NARID must be corrected. The name CV_SM$ENAM defined in the name array structure SM$ENAM is not name array head.
SEVERITY	Error
Author	M. Richter

NARIDOUTRANGE

MESSAGE	A name index of the Data Element '!AS' is out of range.

RECOVERY	The argument 'p_sm$enam' of the procedure call M$NARID must be corrected. An index of the name CV_SM$ENAM is defined smaller or larger in structure SM$ENAM than in the Data Element Directory.
SEVERITY	Error
Author	M. Richter

NARIDWRONGDIM

MESSAGE	The Data Element name array 'name' is has another dimensionality than defined in structure SM$ENAM.
RECOVERY	The argument 'p_sm$enam' of the procedure call M$NARID must be corrected. The dimension L_SM$ENAM_DIM defined in the name array structure SM$ENAM is not the same as that of the name array.
SEVERITY	Error
Author	M. Richter

NARRLIMNOMATCH

MESSAGE	<Module>: The Name Array limits for source Data Element <Data Element name(limits)> and destination Data Element <Data Element name(limits)> doesn't match.
RECOVERY	The specified source DE and destination DE name array limits are not in the same range. Specify a another Data Element or use a another command to copy this Data Element.
SEVERITY	Error
Author	T. KROLL

NDNOTFOUND

MESSAGE	Name 'NAME' to delete not found
RECOVERY	Correct the spelling of the entry name
SEVERITY	ERROR
Author	M. Richter

NEWDIRTOOSMALL

MESSAGE	The Directory 'dir' in the Data Base 'base' can not be enlarged, since the new number of entries 'new' is smaller than the old one 'old'.
RECOVERY	A Directory in a Data Base can only be renewed (enlarged) if the new number of entries is larger than the old one. Increase the number of new entries in the procedure call.
SEVERITY	Error
Author	M. Richter

NGILLINDEX

MESSAGE	Illegal index of Directory entry 'index'
RECOVERY	Correct the input argument l_index to be in the range of the Directory
SEVERITY	ERROR
Author	M. Richter

NGNOENTRY

MESSAGE	Index 'index' points to a Directory entry not in use
RECOVERY	Correct the input argument l_index to be in the range of the Directory
SEVERITY	ERROR
Author	M. Richter

NGNONAME

MESSAGE	Index 'ind' points to a Directory entry with no name
RECOVERY	Correct the input argument l_index to be in the range of the Directory
SEVERITY	ERROR

Author	M. Richter

NINAMEAMBIG

MESSAGE	Name 'name' to insert in Directory was found already or is ambiguous
RECOVERY	Change name.
SEVERITY	Error
Author	M. Richter

NINOTFOUND

MESSAGE	Name 'name' to insert not found
RECOVERY	Change name.
SEVERITY	Error
Author	M. Richter

NLNOMORENAMES

MESSAGE	There are no more names in the Directory <directory>
RECOVERY	In a wildcard operation no more names are found
SEVERITY	Error
Author	M. Richter

NOALLSPECIFY

MESSAGE	<Module>: Source Data Element <Data Element name> and destinantion Data Element <Data Element name> exist as a name array and switch /ALL is not set.
RECOVERY	Set switch /ALL or specify another destination Data Element .
SEVERITY	Error
Author	T. KROLL

NOARRAY

MESSAGE	<Module>: Input Member 'member' does not exist as an array in Type Descriptor.
RECOVERY	Try again with correct input member
SEVERITY	Error
Author	T. Kroll

NOARRAYSPECIFY

MESSAGE	<Module>: Input Member 'member' exist as an array in Type Descriptor.
RECOVERY	Specify for Input Member correct array dimensions.
SEVERITY	Error
Author	T. Kroll

NOCONFIRMSPECIFY

MESSAGE	<Module>: Destination Data Element <Data Element name> doesn't exist and switch /CONFIRM is not set.
RECOVERY	Set switch /CONFIRM or specify another destination Data Element .
SEVERITY	Error
Author	T. KROLL

NOMEMBERFOUND

MESSAGE	'Module' Input Member 'parsed member name' doesn't match.
RECOVERY	Change input member name.
SEVERITY	Error
Author	T. Kroll

NOMOREDB

MESSAGE	Maximum number of Data Bases: '<number>' exceeded.
RECOVERY	This is a severe problem. The maximum number of Data Bases is a relative hard limit. If You really need more, contact the GOOSY manager. A solution is possible but requires a new GOOSY release.
SEVERITY	ERROR
Author	H.G.essel

NOMOREDYN

MESSAGE	<module>: All # slots of local dynamic list directory are used
RECOVERY	# is a GOOSY system generation parameter. It can be increased only by a major update. Contact the GOOSY crew, if You really need more.
SEVERITY	ERROR
Author	H.G.Essel

NOTYETIMPL

MESSAGE	Not yet implemented: specification line.
RECOVERY	None.
SEVERITY	ERROR
Author	H.G.Essel

NOUNMAPADIR

MESSAGE	The Area to delete with the index 'index' of Data Base 'base name' is the Area Directory itself. It cannot be detached (unmapped).
RECOVERY	Calling argument of M$DLAR 'l_area_index' must be corrected. This Area is the Area Directory. It must be copied to a new Area before it is allowed to delete the Area of the Area Directory.

SEVERITY	Error
Author	M. Richter

NOVALIDCOPY

MESSAGE	<Module>: Copy from or to a Structuremember is not a valid copy type.
RECOVERY	Copy full Data Element or use COPY MEMBER command.
SEVERITY	Error
Author	T. KROLL

NOVALIDCOPYTYPE

MESSAGE	<Module>: The specified source Data Element <Data Element name> or destination Data Element <Data Element name> Type is not valid for copy of Data Elements.
RECOVERY	The specified source DE or destination DE are defined by a type which is not valid for a copy Data Element command.
SEVERITY	Error
Author	T. KROLL

NOVALIDNODE

MESSAGE	Copy of an DE to another node is not implemented.
RECOVERY	Specify same node for source and target Data Element.
SEVERITY	Error
Author	T. KROLL

NOVALIDREFERVALUE

MESSAGE	<Module>: The Integer value 'value' is not a valid value to specify the string length of member 'member'.
RECOVERY	Specify for Member correct refer value.

SEVERITY	Error
Author	T. Kroll

NOVALIDSRCVALUE

MESSAGE	<Module>: The Value <value> of source Data Element member <DE name> is not valid to be converted to the target Data Element member <DE name> data type <data type>.
RECOVERY	Correct source or target Data Element member.
SEVERITY	Error
Author	T. KROLL

NOVALINP

MESSAGE	<Module>: The Input 'value' has not the requested datatype defined by type descriptor.
RECOVERY	Specify for input value correct datatyp.
SEVERITY	Error
Author	T. Kroll

NSNOTFOUND

MESSAGE	Name 'name' searched not found
RECOVERY	Change name.
SEVERITY	Error
Author	M. Richter

PLDIRNOTMAP

MESSAGE	The pool directory of data base <base> is not attached (mapped)
RECOVERY	The data base must be attached, before any other actions can be done.

SEVERITY ERROR

Author H.G.Essel

POOLNO

MESSAGE The Pool 'pool' of Data Base 'base'
  is not attached (mapped) with write access

RECOVERY The Pool was not attached with write access. M$ATPO was called with read access only.

SEVERITY Error

Author M. Richter

POOLNOTMAP

MESSAGE <module>: Pool '<pool>' of data base 'base'
  is not attached (mapped)

RECOVERY The pool must be attached, before any other actions can be done.

SEVERITY ERROR

Author H.G.Essel

QENUMNOTEQUAL

MESSAGE <Module>: The number of queued source Data Element <Data Element name> and destination Data Element <Data Element name> are not equal.

RECOVERY For copying a queued Data Element with the switch /ALL the number of queued Data Elements for source and destination Data Elements must be the same. Specify a another Data Element or use a another command to copy this Data Element.

SEVERITY Error

Author T. KROLL

REFOBJILLEDATYP

MESSAGE	<Module> :Refer object <name> has wrong datatyp <wrong datatyp>.
RECOVERY	Correct datatyp for refer obj in type descriptor defining structure.
SEVERITY	Error
Author	T. Kroll

REFOBJMISMATCH

MESSAGE	No variable found for refer object 'ref obj'
RECOVERY	Correct type descriptor defining structure
SEVERITY	Error
Author	T. Kroll

REPLNOTSET

MESSAGE	<Module>: Target Data Element <data element name> exists already in Data Base <data base name> and switch replace is not set.
RECOVERY	Set switch /REPLACE or specify another target Data element.
SEVERITY	Error
Author	T. KROLL

SECFILELOCK

MESSAGE	The Data Base section file 'db-file' is locked. It might be in use on another cluster node.
RECOVERY	The Data Base section file can be in use on one VAX cluster node at a time, only! To use it on a second node, make either a delete Global Section (DELGS) on the node currently using the file or copy the file. In case of a copy the Data Base will not be consistent between the two nodes!

SEVERITY	Error
Author	M. Richter

SHHBATT

MESSAGE	The Data Base 'db' is attached already. The Home Block of a detached Data Base could be shown, only. Use another DM show command with the /NOKEEP_MAP switch to detach the Data Base.
RECOVERY	Use another DM show command with the /NOKEEP_MAP switch to detach the Data Base. Then show the Home Block again.
SEVERITY	Error
Author	M. Richter

SUBRANGE

MESSAGE	<Module>: The Integer value 'value' does not lie in the range 'defined value' for member 'member in Type Descriptor'.
RECOVERY	Specify for Input Member correct array limits.
SEVERITY	Error
Author	T. Kroll

TOOLARGE

MESSAGE	<Module>: Type descriptor too large for destination.
RECOVERY	Calculate the length of a type descriptor by M$TYLEN
SEVERITY	ERROR
Author	H.Essel

TOOMANYSUB

MESSAGE	<Module>: 'member'has been referenced with too many subscripts.
RECOVERY	Specify for Input Member correct array dimensions.

SEVERITY	Error
Author	T. Kroll

TYDIRNOTMAP

MESSAGE	The type directory of Data Base 'base' is not attached (mapped)
RECOVERY	The Data Base must be attached, before any other actions can be done.
SEVERITY	ERROR
Author	H.G.Essel

TYD_INVFULLREFOBJ

MESSAGE	qualified refer object is incorrect, specified prefix '!AS' is not equal to the valid prefix '!AS'.
RECOVERY	modify input file or string and try again.
SEVERITY	Error
Author	K. Winkelmann

TYD_INVJMP

MESSAGE	incorrect jump in level number in input structure, jump from !AS to !AS is not allowed.
RECOVERY	modify input file or string and try again.
SEVERITY	Error
Author	K. Winkelmann

TYD_INVLVL

MESSAGE	invalid level number in strucure: 'l_level_current' level must be between 2 and 15
RECOVERY	modify input file or string and try again.

SEVERITY	Error
Author	K. Winkelmann

TYD_INVVARNAME

MESSAGE	invalid name of variable 'name'
RECOVERY	modify input file or string and try again.
SEVERITY	Error
Author	K. Winkelmann

TYD_NOINPUT

MESSAGE	no input specified, neither file nor string, no type descriptor generated
RECOVERY	specify input parameters.
SEVERITY	Error
Author	K. Winkelmann

TYD_PARS

MESSAGE	error during parsing the input structure for creating the type descriptor>
RECOVERY	see previous output of the parser (including information about the spot where the error occurred) for further details .
SEVERITY	Error
Author	K. Winkelmann

TYD_QUALREF

MESSAGE	qualified refer object given !AS, but prefixes !AS will be ignored
RECOVERY	no recovery necessary. Note however that unqualified refer object names have to be unique.
SEVERITY	Warning
Author	K. Winkelmann

TYD_STRINGOVFL

MESSAGE	built name 'string' is longer than the allowed length of 'stringlength'
RECOVERY	modify the input structure and try again
SEVERITY	Error
Author	K. Winkelmann

TYPDESNOTEQUAL

MESSAGE	The Typdescriptor for 'de-source' and 'de-target' are not equal.
RECOVERY	Correct the mismatching type descriptor.
SEVERITY	Error
Author	T. KROLL

TYPENOTMATCH

MESSAGE	<Modul>: The Data Element type 'type' doesn't match with the requested type 'req_type'.
RECOVERY	Check that the Data element has an correct type for the wanted operation.
SEVERITY	ERROR
Author	T. Kroll

UMILLPOINT

MESSAGE	Pointers of region to unmap are NULL. No unmap possible.
RECOVERY	Error in arguments of M$UMDB. Correct calling procedure.
SEVERITY	Error
Author	M. Richter

GOOGIP

Prefix	XGIP_
Object name	GOOMSGLIB(XGIP)
File name	GOO$MSG:XGIP.MSG
Dataset	RZ88.XGIP.MSG
Identifier	V01.00A
Facility	107

ARIT_E

MESSAGE	Error in evaluation of arithmetic expression: !AS
RECOVERY	modify expression and try again
SEVERITY	ERROR
Author	W.Kynast

ERRMSG

MESSAGE	Error: '!AS'
RECOVERY	-
SEVERITY	ERROR
Author	H. Grein

ERRMSG

MESSAGE	Warning: '!AS'
RECOVERY	-

SEVERITY	WARNING
Author	H. Grein

ERRMSG

MESSAGE	Info: '!AS'
RECOVERY	-
SEVERITY	INFO
Author	H. Grein

ERRMSG

MESSAGE	GIPSY system initialisation failed
RECOVERY	Debug memory allocation system
SEVERITY	FATAL
Author	H. Grein

ERROR

MESSAGE	Function not completed
RECOVERY	look for the reason on error stack
SEVERITY	ERROR
Author	H. Grein

FATAL

MESSAGE	Information
RECOVERY	look for the reason on error stack
SEVERITY	INFO
Author	H. Grein

FATAL

MESSAGE	Warning
RECOVERY	look for the reason on error stack
SEVERITY	WARNING
Author	H. Grein

FATAL

MESSAGE	Unrecoverable error
RECOVERY	look for the reason on error stack
SEVERITY	FATAL
Author	H. Grein

IEOF

MESSAGE	EOF found in included module
RECOVERY	none
SEVERITY	INFO
Author	H. Grein

IFNF

MESSAGE	Include file '!AS' not found
RECOVERY	none
SEVERITY	ERROR
Author	H. Grein

ILNF

MESSAGE	Library '!AS' to include module '!AS' not found

RECOVERY	none
SEVERITY	ERROR
Author	H. Grein

IMNF

MESSAGE	Module '!AS' in library '!AS' not found
RECOVERY	none
SEVERITY	ERROR
Author	H. Grein

INVINC

MESSAGE	Invalid include module specification
RECOVERY	none
SEVERITY	ERROR
Author	H. Grein

ISEMPTY

MESSAGE	Include stack is empty
RECOVERY	Continue input by prompter
SEVERITY	INFO
Author	H. Grein

PARS_ESYM

MESSAGE	endsymbol table overflow occurred when parser <parser> was invoked with <string>
RECOVERY	tell system group to inclrease implementation stack size
SEVERITY	ERROR

Author	K.Winkelmann

PARS_PDL

MESSAGE	pushdownlist overflow occurred when parser <parser> was invoked with <string>
RECOVERY	tell system group to inclrease implementation stack size
SEVERITY	ERROR
Author	K.Winkelmann

PARS_SYNTAX

MESSAGE	Syntax error occurred when parser !AS was invoked with !AS
RECOVERY	use correct syntax , look at the error locator
SEVERITY	ERROR
Author	K.Winkelmann

PARS_SYNTAX

MESSAGE	table for rules overflowed when parser <parser> was invoked with <string>
RECOVERY	tell system group to increase the implementation size
SEVERITY	ERROR
Author	K.Winkelmann

STCKOVFL

MESSAGE	Overflow in stack '!AS'
RECOVERY	none
SEVERITY	INFO
Author	H. Grein

SUCCESS

MESSAGE	Function successfully completed
RECOVERY	none
SEVERITY	SUCCESS
Author	H. Grein

TABOVFL

MESSAGE	Overflow in table '!AS'
RECOVERY	none
SEVERITY	INFO
Author	H. Grein

UNKNWN

MESSAGE	Unidentified status from lower level module
RECOVERY	This message never should appear
SEVERITY	ERROR
Author	H. Grein

GOOIO

Prefix	XIO_
Object name	GOOMSGLIB(XIO)
File name	XIO.MSG
Dataset	RZ88.XIO.MSG
Identifier	V01.00A
Facility #	103

BADBDO

MESSAGE	Inconsistent MBD channel program <file_name>
RECOVERY	Signals that the file <file_name> contains a MBD module with inconsistent load address and/or length information. This is probably due to a misstyped MBD program header in the source text. Correct the source, recompile and rerun the program or command.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACA

MESSAGE	Bad CAMAC subaddress <A> specified
RECOVERY	A subaddress less than 0 or greater than 15 was used. Check and correct the subaddress.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACB

MESSAGE Bad CAMAC branch number <B> specified

RECOVERY A branch number less than 0 or greater than 63 was used.
Check and correct the branch number.

SEVERITY ERROR

Author Walter F.J. Mueller

BADCAMACC

MESSAGE Bad CAMAC crate number <C> specified

RECOVERY A crate number less than 1 or greater than 7 was used. Note, that all crate numbers are physical crate numbers in this implementation.
Check and correct the crate number.

SEVERITY ERROR

Author Walter F.J. Mueller

BADCAMACCI

MESSAGE Bad CAMAC channel identifier <CI> specified

RECOVERY A bad channel identifier was specified in a control block of a ESONE CAMAC call. The channel identifier is an index in a channel description table in this implementation and must be either 0 for the default channel or between 1 and 16 for an explicitely specified channel.
NOTE, that a call to I$CDCHN is the only way to get a valid channel identifier.
Check and correct the channel identifier.

SEVERITY ERROR

Author Walter F.J. Mueller

BADCAMACD

MESSAGE Illegal dimension <d> specified in CAMAC call

RECOVERY	An array dimension was inconsistend with the repeat count specified in the control block of a multiple action of block transfer call. Note, that the upper index bound of the external address and function arrays must be greater or equal 1 and the upper index bound of the data array and the Q status array must be greater or equal the repeat count. Note further, that this is checked only for the PL/I entries but not for the ESONE FORTRAN entries. Check and correct the array dimensioning.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACE

MESSAGE	Illegal external address specified in CAMAC call
RECOVERY	An illegal (not generated with CDREG) external address was detected. This is probably the result is a missing initialisation or an illegal attempt to modify an external address. Check and correct the initialisation.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACF

MESSAGE	Bad CAMAC function <F> specified
RECOVERY	A function less than 0 or greater than 31 was used. Check and correct the function.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACL

MESSAGE	Illegal LAM identifier specified in CAMAC call

RECOVERY	A LAM identifier not equal 0 was used in the control block of a multiple action or block transfer call. Note, that in current implentation LAM driven actions are not supported. For future compatibility a LAM identifier of 0 is the only allowed one. Specify 0 for the LAM identifier.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACN

MESSAGE	Bad CAMAC station number <N> specified
RECOVERY	A station number less than 1 or greater than 31 was used. Note, that all station numbers are physical station numbers in this implementation. Note further, that the stations numbers 25 to 31 access registers in the crate controller. Check and correct the station number.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADCAMACR

MESSAGE	Bad CAMAC repeat count <rc> specified
RECOVERY	A repeat count of less than 1 or greater than 4096 has been specified in a control block of a multiple action of block transfer call. Check the proper set-up of the control block. Note, that 4096 is the maximal repeat count permitted in this implementation.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADEVENT

MESSAGE	Bad event found: type=<number>, subtype=<number>, buffer index=<index>.

RECOVERY An event has been found with a corrupt data length field. The buffer may be processed further, but consecutive errors may occur. The length fields are not used to parse the events, but if they are not correct, something else may be wrong. The buffer index is the word index of the buffer data array (without header).

SEVERITY WARNING

Author H.G.Essel

BADJCDRV

MESSAGE <Name> is not a valid reference to the MBD driver

RECOVERY Signals, that <Name> is not a legal MBD root device name.

Note, that a MBD device specification <name> is used in the following way:
The three logical names

    <name>A, <name>B, <name>C

are translated. The equivalence name must refer to a controler/unit of the JD driver which is the currently used MBD driver. The default MBD devive specification is "MBD", therefore check whether the logical names

    MBDA,MBDB and MBDC

exist and translate in "JDx0:".

SEVERITY ERROR

Author Walter F.J. Mueller

BADMBDEVENT

MESSAGE Bad MBD event found:
<reason>.

RECOVERY An event has been found with an invalid crate number. Check the J11 readout programs.

SEVERITY WARNING

Author H.G.Essel

BADMBDEXEC

MESSAGE	Invalid MBD executive found in <file>, base is <base>
RECOVERY	An attempt was made, to load a MBD executive with a nonzero base address. Check source program, correct program header.
SEVERITY	ERROR
Author	Walter F.J. Mueller

BADMBDVFY

MESSAGE	<Number> mismatches during MBD load verification
RECOVERY	Signals, that the read verification after a MBD channel program load failed. The read check found <Number> words differences to the loaded program. This indicates in most cases a MBD memory error and is a SEVERE hardware failure. It should be corrected before any further accesses to the MBD are tried. NOTE: A MBD memory fault may cause a VAX system crash !
SEVERITY	ERROR
Author	Walter F.J. Mueller

BUFFERFULL

MESSAGE	List mode buffer full.
RECOVERY	This message is used to signal that an event pack routine cannot copy the current event into the output buffer. A new buffer must be provided.
SEVERITY	WARNING
Author	H.G.Essel

BUGCHK

MESSAGE	BUGCHECK. Internal consistency check, syndrome: <syndrome>

RECOVERY	Signals, that a procedure detected a fatal internal consistency error which is unextected and unrecoverable. The syndrome text specifies the bugcheck reason, the interpretation depends on the procedure. Ask the author to correct the problem.
SEVERITY	ER
Author	Walter F.J. Mueller

CAMNOXNOQ

MESSAGE	CAMAC action completed with no X and no Q
RECOVERY	This indicates, that at least one of the CAMAC actions of a multiple action call (or the one of a single action call) has completed with no X and no Q response. NOTE: This condition probably indicates that an empty CAMAC slot or a damaged module has been accessed. Check the Q status array to identify the actions with no X response in a multiple action call.
SEVERITY	SUCCESS
Author	Walter F.J. Mueller

CAMNOXQ

MESSAGE	CAMAC action completed with no X but with Q (STRANGE)
RECOVERY	This indicates, that all CAMAC actions of a multiple action call (or the one of a single action call) have been completed with a Q response but that at least one action (or the one of a single action call) as completed with no X response. NOTE: This is a somewhat strange condition. All CAMAC modules should respond with X if they acknowledge a function with a Q response. This condition therefore probably indicates a damaged CAMAC module, dataway or crate controler ! Check the Q status array to identify the actions with no X response in a multiple action call.
SEVERITY	SUCCESS
Author	Walter F.J. Mueller

CAMXNOQ

MESSAGE	CAMAC action completed with X but no Q
RECOVERY	This indicates, that all CAMAC actions of a multiple action call (or the one of a single action call) have been completed with a X response but that at least one action (or the one of a single action call) as completed with no Q response. Check the Q status array to identify the actions with no Q response in a multiple action call.
SEVERITY	SUCCESS
Author	Walter F.J. Mueller

CAMXQ

MESSAGE	CAMAC action completed with X and Q
RECOVERY	This indicates, that all CAMAC actions of a multiple action call (or the one of a single action call) have been completed with a X and a Q response.
SEVERITY	SUCCESS
Author	Walter F.J. Mueller

ERROR

MESSAGE	Error: <syndrom>
RECOVERY	No recovery possible. Signals, that a general error condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	ERROR
Author	Walter F.J. Mueller

EVTLOOPER

MESSAGE	An error occured in buffer #, event #, buffer event #.

RECOVERY	This error message is added in the event loop, if the unpack or analysis routines reported some error. It specifies the buffer and event number where the error occured. Buffer event is the event number in the current buffer.
SEVERITY	WARNING
Author	H.G.Essel

EVTOOBIG

MESSAGE	Event is too big for buffer.
RECOVERY	This error indicates that an event could not be copied into a buffer because it is too big. There is no simple recovery for that case. The buffersize must be enlarged. Contact the GOOSY group.
SEVERITY	WARNING
Author	H.G.Essel

EVTOOBIG

MESSAGE	Event data element is too small for event of size #.
RECOVERY	This error indicates that an event could not be copied into the event data element, because it is too big. Check the size of the SA$MBD structure.
SEVERITY	WARNING
Author	H.G.Essel

FATAL

MESSAGE	Fatal Error: <syndrom>
RECOVERY	No recovery possible. Signals, that a general fatal condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	FATAL
Author	Walter F.J. Mueller

FILENOTOPEN

MESSAGE	File not open
RECOVERY	The file must be opened first.
SEVERITY	WARNING
Author	H.G.Essel

FILESTILLOPEN

MESSAGE	File still open
RECOVERY	The file must be closed first.
SEVERITY	WARNING
Author	H.G.Essel

ILLBDO

MESSAGE	Illegal record format for MBD object module <file_name>
RECOVERY	Signals that the file <file_name> contains a record which is illegal for a MBD object module. Probably the file is a not an object module but a source program ect.. Rerun the program/command with the correct file specification.
SEVERITY	ERROR
Author	Walter F.J. Mueller

ILLMODE

MESSAGE	An illegal mode <number> has been specified
RECOVERY	Signals, that an illegal mode was specified in an argument list. Correct the calling program.
SEVERITY	ER
Author	Walter F.J. Mueller

INFILSTOP

MESSAGE	Input from file already stopped.
RECOVERY	This message means that a STOP INPUT FILE had no effect, because the file input channel was not open.
SEVERITY	WARNING
Author	H.G.Essel

INFORM

MESSAGE	Information: <syndrom>
RECOVERY	No recovery required. Signals, that a general informative condition occured during the execution of the module.
SEVERITY	INFORMATIVE
Author	Walter F.J. Mueller

INMBXSTOP

MESSAGE	Input from mailbox already stopped.
RECOVERY	This message means that a STOP INPUT MAIL had no effect, because the mailbox channel was not open.
SEVERITY	WARNING
Author	H.G.Essel

INNETSTOP

MESSAGE	Input from DECnet already stopped.
RECOVERY	This message means that a STOP INPUT NET had no effect, because the DECnet channel was not open.
SEVERITY	WARNING
Author	H.G.Essel

ISMBDEXEC

MESSAGE	The file <file> may contain a MBD executive
RECOVERY	An attempt was made, to load a MBD executive as a channel program. This may be due to a misstyed MBD program header or due to a misstyped file name. Note, that the channel 7 is reserved for the executive and cannot be used for application programs.
SEVERITY	ERROR
Author	Walter F.J. Mueller

J11TABERR

MESSAGE	J11 configuration table error in file <file>, line <error line> Item '<item>', value '<value>'
RECOVERY	Look in the description of the LOAD J11 command for the syntax of the CAMAC description. Correct file and retry.
SEVERITY	ERROR
Author	H.G.Essel

LOADERROR

MESSAGE	Error loading module <module> in sharable image <image>
RECOVERY	The module must be linked in the sharable image. Check the sharable image load map.
SEVERITY	ERROR
Author	H.G.Essel

MOUNTERR

MESSAGE	Tape mount error: <reason>

RECOVERY	The tape was mounted in a wrong way. Probably the blocksize was not correct. The blocksize must be a multiple of the GOOSY buffer size. The GOOSY command MOUNT TAPE provides proper defaults.
SEVERITY	ERROR
Author	H.G.Essel

NOEVENTDE

MESSAGE	There is no data element to copy input event
RECOVERY	The unpack routine has no valid pointer to a GOOSY data element to copy en event. There must have been some error during the initialization of the unpack routine. If the signaling routine was X$UPJ11, X$UPCMP or X$UPEVT, please contact the GOOSY system group.
SEVERITY	ERROR
Author	H.G.Essel

NOMOREEVENT

MESSAGE	No more events in buffer
RECOVERY	The unpack routine returns this error to signal that there are no more events to process in the current buffer. A new buffer must be provided.
SEVERITY	WARNING
Author	H.G.Essel

NOOUTPUT

MESSAGE	No event for output
RECOVERY	Used by analysis routines to signal that no event should be written to output buffer.
SEVERITY	WARNING
Author	H.G.Essel

NOTMBDEXEC

MESSAGE	Invalid MBD executive found in <file>, channel is <channel>
RECOVERY	An attempt was made, to load a MBD executive for a channel different than 7. Check source program, correct program header.
SEVERITY	ERROR
Author	Walter F.J. Mueller

OUTNOSTART

MESSAGE	Analysis output not started.
RECOVERY	Analysis output must be started first.
SEVERITY	WARNING
Author	H.G.Essel

OUTSTART

MESSAGE	Analysis output already started.
RECOVERY	Analysis output must be stopped first.
SEVERITY	WARNING
Author	H.G.Essel

SKIPEVENT

MESSAGE	Skip event
RECOVERY	This message is used by event loop routines to skip the following event processing.
SEVERITY	WARNING
Author	H.G.Essel

STOPINPUT

MESSAGE	Stop input.
RECOVERY	This message code is used to signal the caller to stop the data input.
SEVERITY	WARNING
Author	H.G.Essel

STOPOUTPUT

MESSAGE	Stop output.
RECOVERY	This message code is used to signal the caller to stop the data output.
SEVERITY	WARNING
Author	H.G.Essel

SUCCESS

MESSAGE	Success: <syndrom>
RECOVERY	No recovery required. Signals only, that a module of the I/O subsystem completed successfully.
SEVERITY	SUCCESS
Author	Walter F.J. Mueller

UNKNBUF

MESSAGE	Unknown buffer type <number>, subtype <number>
RECOVERY	The unpack routine used cannot unpack the buffer, because the type does not match. Check the type of buffers and use the correct routine.
SEVERITY	WARNING
Author	H.G.Essel

UNKNEVT

MESSAGE	Unknown event type <number>, subtype <number> at buffer data index <index>.
RECOVERY	The unpack routine used cannot unpack the buffer, because the event type does not match. Check the type of the events and use the correct routine. The buffer index is the word index of the buffer data array (without header).
SEVERITY	WARNING
Author	H.G.Essel

VMEDESTERR

MESSAGE	VME destination processor error: <reason>
RECOVERY	The processor specification for a command was not valid.
SEVERITY	ERROR
Author	H.G.Essel

VMETABERR

MESSAGE	VME configuration table error in file <file>, line <error line> Item '<item>', value '<value>'
RECOVERY	Look in the description of the VME system for the syntax of the VME description. Correct file and retry.
SEVERITY	ERROR
Author	H.G.Essel

WARNING

MESSAGE	Warning: <syndrom>

RECOVERY	No recovery possible. Signals, that a general warning condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	WARNING
Author	Walter F.J. Mueller

GOONET

Prefix	XNET_
Object name	GOOMSGLIB(XNET)
File name	XNET.MSG
Dataset	RZ88.XNET.MSG
Identifier	V01.00A
Facility	106

ACKNDONE

MESSAGE	Transaction acknowledge already done
RECOVERY	Signals an attemp to send an acknowledge message for an already acknowledged transaction. Note, that a transaction can be acknowledged only once and any further acknowlege attempts are discarted. Check the logic of the message/transaction handler.
SEVERITY	ER
Author	Walter F.J. Mueller

ACKNDROPPED

MESSAGE	Transaction acknowlegde message unexpected, dropped
RECOVERY	Signals, that a transaction expected a status acknowledge but received a full acknowledge message. The unexpected message was dropped. Check, whether the transaction handler on the other link end works as assumed.
SEVERITY	ER
Author	Walter F.J. Mueller

ACKNMISSED

MESSAGE	Transaction acknowlegde message expected and missed
RECOVERY	Signals, that a transaction expected an acknowledge message but received a simple status acknowledge. Check, whether the transaction handler on the other link end works as assumed.
SEVERITY	ER
Author	Walter F.J. Mueller

BADDTYPE

MESSAGE	Bad data type/class in a RPCG call: <syndrome>
RECOVERY	Signals, that a 'general mode' RPC procedure was called with an illegal data type (N$RPTVG), an unsupported data type/class descriptor (N$RPTDG) an illegal reference type or an illegal combination of data and reference type. Check data type passed, use only supported data types.
SEVERITY	ER
Author	Walter F.J. Mueller d call sequence for RPCG procedure: !/!AS" /FAO=1

BADGOOSYNAME

MESSAGE	Bad GOOSY name component <name>
RECOVERY	Signals, that the indicated part of a GOOSY object name has either an illegal length or contains illegal characters. Note, that the name components should not be longer than 4 characters and should not contain any special characters.
SEVERITY	ER
Author	Walter F.J. Mueller

BADLINKNAME

MESSAGE	Syntax error on link name <link_name>, <syndrome>

RECOVERY The specified link name has a syntax error described by <syndrome>. Note, that the link destination must have the format:

[network%][node::]object(component)

The link destination may either specified directly or as a logical name. Note, that the error message contains the equivalence string if a logical name has been specified.
Check the naming, redefine the logical name.

SEVERITY ER

Author Walter F.J. Mueller

BADLINKOPEN

MESSAGE Failed to open a static link to <link_destination>

RECOVERY Signals, that the open of a static link failed. Check the previous message in the error stack for the detailed reason.
NOTE, that this error is only reported for synchronous link opens. The link I/O status (SN$LCB_LIOSB) has to be checked in case of an asynchronous open.

SEVERITY ER

Author Walter F.J. Mueller

BADOBJNAME

MESSAGE Bad network object name <name>

RECOVERY Signals, that the network object name specified in a N$PDCLO call is of illegal format, e.g. of zero length or longer than 12 characters.
Use another object name.

SEVERITY ER

Author Walter F.J. Mueller

BADRPCGSEQ

MESSAGE Bad call sequence for RPCG procedure: <syndrome>

RECOVERY	Signals, that a 'general mode' RPC procedure was called out of context or sequence. There are quite a few rule to obey, the <symdrome> specifies which was violated. Check arguments and program logic.
SEVERITY	ER
Author	Walter F.J. Mueller

BADSTRUCSIZE

MESSAGE	Wrong structure size in a RPC call, found <f>, expected <e>
RECOVERY	Signals, that a RPC request was done with an unexpected or illegal structure size or that a RPC entry returned an illegal size for the output structure. Check call (N$RSEXS/N$RBEXS) in client or logic of handler.
SEVERITY	ER
Author	Walter F.J. Mueller

BUGCHK

MESSAGE	BUGCHECK. Internal consistency check, syndrome: <syndrome>
RECOVERY	Signals, that a procedure detected a fatal internal consistency error which is unexpected and unrecoverable. The syndrome text specifies the bugcheck reason, the interpretation depends on the procedure. Ask the author to correct the problem.
SEVERITY	ER
Author	Walter F.J. Mueller

COMPUNKNOWN

MESSAGE	Component <name> is unknown in object
RECOVERY	Signals, that the specified component has not yet been declared in the object. Note, that components must be declared with N$PDCLC before they can be referenced.
SEVERITY	ER

Author	Walter F.J. Mueller

DROPMSG

MESSAGE	Message dropped from source <link_destination> first data: <begin of message in ASCII>
RECOVERY	Signals, that a message received on the indicated link could not be handled and was dropped. The first 48 bytes of the message data part are displayed in ASCII, all nonprintable characters are replaced by a colon (.). This message is primarily generated by the message drop handler N$LDRPM which is in most cases the default message handler of a component. This results in dropping all messages for which no explicit message handlers have been declared. Check, whether the link partner runs the right protocol and whether it is the right component.
SEVERITY	ER
Author	Walter F.J. Mueller

DUMMY

MESSAGE	General purpose dummy N$DUMMY executed
RECOVERY	Signals, that the procedure N$DUMMY was called. Note, that N$DUMMY is the only procedure which returns this status. A check on XNET_DUMMY may be used to determine, whether N$DUMMY was called in a handling procedure.
SEVERITY	SU
Author	Walter F.J. Mueller

DUPLNAME

MESSAGE	Duplicate component name <name>
RECOVERY	Signals an attempt to declare a component with a name of an already existing component. Check the naming of components.
SEVERITY	ER

Author Walter F.J. Mueller

DUPLOBJNAME

MESSAGE Duplicate network object name <name>

RECOVERY Signals, that the network object name specified in a N$PDCLO call is already in use.
Use another object name.

SEVERITY ER

Author Walter F.J. Mueller

ENVALREX

MESSAGE Environment env already exists.
Use different name!

RECOVERY The environment names must be unique. The SHOW USER command shows GOOSY processes wich start with GN_env...

SEVERITY WARNING

Author H.G.Essel

ERROR

MESSAGE Error condition

RECOVERY No recovery possible. Signals, that a general error condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.

SEVERITY ER

Author Walter F.J. Mueller

FATAL

MESSAGE Fatal condition

RECOVERY	No recovery possible. Signals, that a general fatal condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	FA
Author	Walter F.J. Mueller

ILLDIM

MESSAGE	Illegal or inconsistent array dimension
RECOVERY	Signals, that either an array passed to an entry had an illegal lower or upper dimension bound or that multiple arrays had inconsistend bounds (e.g. differing bounds when equal bounds are required.) Check call and declaration of passed arguments.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLENTIND

MESSAGE	Illegal entry point index <index> in RPC
RECOVERY	Signals, that a nonpositive or too large entry point index was specified in a RPC (remote procedure call) operation. Check, whether the correct link, message type, subtype and entry point index combination was used.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLMODE

MESSAGE	An illegal mode <number> has been specified
RECOVERY	Signals, that an illegal mode was specified in an argument list. Correct the calling program.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLMSGSIZ

MESSAGE	Illegal message size <size>
RECOVERY	Signals, that a negative or too large message size was specified in a GNA operation like allocation of free of a message control block (MCB). Check the calculation of the message length.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLNULLPTR

MESSAGE	Illegal NULL pointer specified in argument list
RECOVERY	Signals, that NULL pointer was specified where a reference to a GNA control structure like a LCB or MCB was expected. Note, that the default pointers of the GNA control structures are all declared with INIT(NULL). A missing initialisation of such a pointer therefore results in either a access violation or if passed to GNA procedure in this status code.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLNULLSTR

MESSAGE	Illegal null string specified in argument list
RECOVERY	Signals, that a zero length string has been specified in an argument list where a non-zero string is expected. Check the calling procedure.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLRECSIZE

MESSAGE	Illegal record size <size>

RECOVERY	Signals, that a too small or too large record size has been specified.
SEVERITY	ER
Author	Walter F.J. Mueller

ILLWLDCHR

MESSAGE	Only full wildcard supported for field <field>
RECOVERY	Signals, that the procedure supports only a full wildcard for the given name and that a more general wildcard construct like "xxx*" has been specified. Reenter the command.
SEVERITY	WA
Author	Walter F.J. Mueller

INFORM

MESSAGE	Informational condition
RECOVERY	No recovery required. Signals, that a general informative condition occured during the execution of the module.
SEVERITY	IN
Author	Walter F.J. Mueller

INITDONE

MESSAGE	GNA initialisation already done, redundant call of N$xINIT
RECOVERY	Signals that N$LINIT or N$PINIT has been called more than once. Note, that one of these procedures (in general N$LINIT) must be called once during the process initialisation. All later calls are ignored and result in this informative status. Inspect process initialization and remove the superfluous calls of N$LINIT or N$PINIT.
SEVERITY	IN
Author	Walter F.J. Mueller

INITIMEOUT

MESSAGE	Timeout during process <prcnam> initialisation
RECOVERY	Signals, that a GNA subprocess creation failed and was aborted after a timeout limit was reached. Possible reasons:
*	The subprocess failed to start or to initialise and logged out.
*	The image started doesn't call N$IDONE (or N$ICHCK directly) to signal the completion of the initialisation phase. Check for preceeding error messages indicating a failure during initialisation or check whether the image calls N$IDONE.
SEVERITY	ER
Author	Walter F.J. Mueller

ISGOOSYPROC

MESSAGE	Process <prcnam> is already a GOOSY process
RECOVERY	Signals, that the creation of a GNA session failed because the job is already a GNA session.
SEVERITY	ER
Author	Walter F.J. Mueller

J11DEADMOD

MESSAGE	Node <node>: CAMAC module at station <N>, subaddress <A>: No X-response.
RECOVERY	Check CAMAC module and retry.
SEVERITY	FATAL
Author	H.Sohlbach

J11ERROR

MESSAGE	Undefined error in J11

RECOVERY	None
SEVERITY	ERROR
Author	H.Sohlbach

J11EVTOOLONG

MESSAGE	Event length exceeds J11 buffer size
RECOVERY	The CAMAC setup should be adjusted.
SEVERITY	WARNING
Author	H.Sohlbach

J11NOMOD

MESSAGE	File <file>: No modules specified.
RECOVERY	Check the configuration file. There must be at least one module specified.
SEVERITY	FATAL
Author	H.Sohlbach

J11UNKNMOD

MESSAGE	Node <node>: module at station <N>, subaddress <A>: not known.
RECOVERY	Check CAMAC module, correct configuration file and retry.
SEVERITY	FATAL
Author	H.Sohlbach

J11UNKNSTAT

MESSAGE	Node <node>: CAMAC status unknown.
RECOVERY	Severe software failure in J11. Submit SPR.
SEVERITY	FATAL

Author	H.Sohlbach

LINKISACTIVE

MESSAGE	Link <link_destination> is active, no inbound connections allowed
RECOVERY	Signals, that an incomming connect request has been detected and rejected for an active link. Note, that active links can have only outbound connections, they must be opened from the active side. Check, whether the right connections have been made.
SEVERITY	ER
Author	Walter F.J. Mueller

LINKISCLOSED

MESSAGE	Link <link_destination> is closed
RECOVERY	Signals, that a link close function like N$PABOL has been called for an already closed link. Note that transport functions return the XNET_LINKNOTOPEN code if they cannot complete because a link is not open.
SEVERITY	ER
Author	Walter F.J. Mueller

LINKISPASSIVE

MESSAGE	Link <link_destination> is passive, no outbound connections allowed
RECOVERY	Signals, that an open function was issued for a passive link. Note, that passive links can have only inbound connections and must be opened from the other side. Check, whether the right connections have been made.
SEVERITY	ER
Author	Walter F.J. Mueller

LINKNOTCLOSED

MESSAGE	Link <link_destination> is not closed

RECOVERY Signals, that any kind of open function has been called for a link which is currently not closed. NOTE: If a link is to be reopened after a link failure it must have been reset with a N$PABOL call before.

SEVERITY ER

Author Walter F.J. Mueller

LINKNOTCONNECTED

MESSAGE Link <link_destination> is not connected

RECOVERY Signals, that a link accept function like N$PACCL has been called for a link which is currently not being initialised. NOTE, that link accept functions can only be used in a connect handler and only for the link currently being handled. This status may also result from a link failure during initialisation.

SEVERITY ER

Author Walter F.J. Mueller

LINKNOTDISPATCHED

MESSAGE Link <link_destination> is not dispatched

RECOVERY Signals, that a transaction or parameter set function has been called and that the link partner is not a dispatched component.
NOTE, that all transaction layer functions need a dispatched partner. Nondispatched components should be an exception anyway.

SEVERITY ER

Author Walter F.J. Mueller

LINKNOTOPEN

MESSAGE Link <link_destination> is not open

RECOVERY Signals, that any kind of transport function has been called for a link which is currently not open. Possible reasons and recoveries:

* Link was never opened. Call N$LAOPL or any other open functions to open it.

*	Link was opened but the initialisation is not yet completed. Check proper synchronisation after an asynchronous link open like N$LAOPL. A link disconnected due to a network failure or due to a failure of the network partner. Wait untill the link is reconnected.
SEVERITY	ER
Author	Walter F.J. Mueller

MCBGUARDCHECK

MESSAGE	An MCB Guard word was overwritten, found: <syndrome>
RECOVERY	Signals, that the guard word at the end of the data part of a MCB was overwritten and indicates, that the program might have overwritten other crucial data too. Check program logic, you might have allocated too small a MCB or underestimated to amount of data written into it.
SEVERITY	ER
Author	Walter F.J. Mueller

MCBQERROR

MESSAGE	An MCB is in a wrong queue, queue state <state>

RECOVERY	Signals, that a message control block was specified for a GNA function while it was still busy with another operation.
	Note, that a message control block (MCB) can be in various queues as indicated by the IN$MCB_QSTATE member but only in one at a time. Any attempt to use an MCB already in a queue in an operation which will put it in another queue results in this error. Look in the descriptions of the various GNA functions about the queue state requirements.
	Currently the MCB queue states are:
	N$_MCBQSTATE_NONE 0;
	N$_MCBQSTATE_PAQ 1;
	N$_MCBQSTATE_LCBI 2;
	N$_MCBQSTATE_LCBO 3;
	N$_MCBQSTATE_LCBTI 4;
	N$_MCBQSTATE_LCBTO 5;

SEVERITY	ER

Author Walter F.J. Mueller

NOINIT

MESSAGE GNA has not been successfully initialized, call N$xINIT

RECOVERY Signals, that a GNA procedure has been called without previous initialisation of the GNA database with a N$PINIT or N$LINIT call. Note that one of these procedures (in general N$LINIT) must be called once during the process initialisation before any other GNA procedure can be executed.
Inspect the process initialisation, add a N$LINIT call or check for errors of the execution of N$LINIT.

SEVERITY ER

Author Walter F.J. Mueller

NOTGOOSYPROC

MESSAGE Process <prcnam> is not a GOOSY process

RECOVERY Signals, that a GNA process control function was issued from a non GNA process.

SEVERITY ER

Author Walter F.J. Mueller

NOTMAINPROC

MESSAGE Action supported only in main process, <prcnam> is none

RECOVERY Signals, that a GNA process control function like spawn of a GNA process has been executed in a subprocess. These functions are only allowed from a main process to avoid improper subprocess nesting.
This error is returned if a CRETE PROCESS or DELETE PROCESS command is given in a SPAWNED process. Attach to the main process and repeat the command.

SEVERITY ER

Author Walter F.J. Mueller

NOTRANSACTION

MESSAGE	A transaction handler was called with a message.
RECOVERY	Signals, that a transaction handler (that is a handler declared with the N$LDCLM option N$M_DCLM_TRANSACTION) was to be called with a simple message. Check logic of link partner, make sure that only transactions are executed for these message types/subtypes.
SEVERITY	ER
Author	Walter F.J. Mueller

PENDING

MESSAGE	Message or transaction still pending
RECOVERY	This is an alternate success status of message and transaction handlers. This and only this status prevents the release of the handled MCB and the return of a status acknowledge message. NOTE, that the MCB has to be freed that an acknowledge message has be send explicitely if this status code is returned by a handler. This however may be done at a later time, e.g. after a FORK DISPATCH.
SEVERITY	SU
Author	Walter F.J. Mueller

QIOFAILURE

MESSAGE	Failed to issue QIO for link <link_destination>
RECOVERY	Signals, that a physical layer function failed to issue a QIO or received an I/O status error.
SEVERITY	ER
Author	Walter F.J. Mueller

RECSHORT

MESSAGE	Protocol error: Input record to short, length: <lgt>

RECOVERY	Signals, that a record with a length shorted than the expected record header length was received. This is an internal consistency error. Check whether physical and logical operations have been mixed for a link. If not, report to the system manager.
SEVERITY	ER
Author	Walter F.J. Mueller

SPWFAILURE

MESSAGE	Failed to start image <image_name> in process <prcnam>
RECOVERY	Signals, that a GNA subprocess creation failed. Check to other error messages to determine why.
SEVERITY	ER
Author	Walter F.J. Mueller

SUCCESS

MESSAGE	Success condition
RECOVERY	No recovery required. Signals only, that a module of the command subsystem completed successfully.
SEVERITY	SU
Author	Walter F.J. Mueller

TRACECLOSE

MESSAGE	Trace file is closed
RECOVERY	Signals, that a trace functions has been activated while the trace file is closed.
SEVERITY	ER
Author	Walter F.J. Mueller

TRACEERROR

MESSAGE	Error while writing to trace file
RECOVERY	Signals, that the trace writer failed to write a trace record to an open trace file.
SEVERITY	ER
Author	Walter F.J. Mueller

UNKNOWNMSG

MESSAGE

Unknown type,subtype = (<type>,<subtype>), data length: <lgt>
  TSN: <tsn>, MODE: <mode>, STATUS: <status>

RECOVERY

Signals, that a message could not be handled because no handler was declared for this type and subtype. The following members of the message control block (MCB) are shown:

    IN$MCB_MSG_TYPE
    IN$MCB_MSG_SUBTYPE
    LN$MCB_DATA_SIZE
    LN$MCB_TSN
    BN$MCB_MODE
    LN$MCB_STAT_ACKN

This message is primarily generated by the message drop handler N$LDRPM which is in most cases the default message handler of a component. This results in dropping all messages for which no explicit message handlers have been declared.
Check, whether the link partner runs the right protocol and whether it is the right component.

SEVERITY ER

Author Walter F.J. Mueller

WARNING

MESSAGE Warning condition

RECOVERY	No recovery possible. Signals, that a general warning condition occured during the execution of the module. This condition should only be used in the test phase of modules to return a status without a more specific diagnostic. If seen, ask the author of the module.
SEVERITY	WA
Author	Walter F.J. Mueller

GOOSYS

Prefix	XSYS_
Object name	GOOMSGLIB(XSYS)
File name	GOO$MSG:XSYS.MSG
Dataset	RZ88.XSYS.MSG
Identifier	V01.00A
Facility	104

ARRBNMAT

MESSAGE	The bounds of !AS do not meet the required range of !AS.
RECOVERY	Check the bound of the array given in the message , correct and try again
SEVERITY	ERROR
Author	K.Winkelmann

EXPIR

MESSAGE	Break due to unauthorized GOOSY activation!
RECOVERY	The GOOSY system was released only for a certain time and the time limit has been reached. Or the GOOSY system has been delivered for a certain processor type only. Or the GOOSY system has been delivered for a certain processor only. Contact the GOOSY managers at GSI.
SEVERITY	ERROR
Author	H.G.Essel

MESINDERR

MESSAGE	The specified error stack index was out of range or the specified error code was not on the stack.
RECOVERY	Check arguments to S$MS...routines. If a certain error has been searched on the stack, this code means that the error was not found.
SEVERITY	WARNING
Author	H.Essel

MESOVFLW

MESSAGE	A full stack has been dumped. Last message is repeated
RECOVERY	If a message stack overflow occurs, all messages including this message are printed. The last message is copied to second message of new stack. This message is copied to first message of new stack
SEVERITY	WARNING
Author	H.E.ESsel

NOPROFINAST

MESSAGE	Not allowed to modify message profile from AST-routine, request ignored
RECOVERY	modify the message profile only from non AST-routines
SEVERITY	WARNING
Author	K.Winkelmann

PROGERR

MESSAGE	Error during program startup!
RECOVERY	A program started in a GOOSY environment terminated with an error. There should be output messages describing the reason.
SEVERITY	ERROR

Author	H.G.Essel

TRACE

MESSAGE	Calling sequence module<-module<-module...
RECOVERY	Trace information
SEVERITY	WARNING
Author	H.G.Essel

TRACEBACK

MESSAGE	Traceback follows, ignore first 2 frames
RECOVERY	No recovery required. Message should appear only with other messages.
SEVERITY	WARNING
Author	W.F.J.Mueller

GOOTEST

Prefix	XTEST_
Object name	GOOMSGLIB(XTEST)
File name	XTEST.MSG
Dataset	RZ88.XTEST.MSG
Identifier	V01.00A
Facility	100

CLUPERR

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

ER

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

ERRCMD

MESSAGE	error in command line '<commandline>'
RECOVERY	You ugly bastard typed complete nonsense !

SEVERITY	WARNING
Author	@

FA

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

IN

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

REPLACE

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

SU

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

TEST

MESSAGE	Test error par1= <parameter>, par2= <parameter>
RECOVERY	This error tests only YOUR intelligence. Go to your manager and get a goody.
SEVERITY	WARNING
Author	@

TEST2

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

TEST3

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

TEST_E

MESSAGE	
RECOVERY	@
SEVERITY	WARNING
Author	@

WA

MESSAGE	

RECOVERY	@
SEVERITY	WARNING
Author	@

GOOUTIL

Prefix	XUTIL_
Object name	GOOLIB(XUTIL)
File name	GOO$MSG:XUTIL.MSG
Dataset	RZ88.XUTIL.MSG
Identifier	V01.00A
Facility	105

COMPERR

MESSAGE	error during compile of specified string or file within a dummy procedure frame
RECOVERY	look at the specific compiler messages, correct the input sring or file and try again
SEVERITY	ERROR
Author	K.Winkelmann

DCLDEB

MESSAGE	<error description>
	Error in DCL line:
	<DCL line>
RECOVERY	Check the DCL command procedure analyzed.
SEVERITY	ERROR
Author	H.G.Essel

EMPSTR

MESSAGE	<module>: Found empty string for <item>
RECOVERY	Check input strings.
SEVERITY	W
Author	H.G.Essel

EOF

MESSAGE	End of File detected
RECOVERY	A read request detected an EOF condition. If this is happens for a command input request, the process should terminate.
SEVERITY	ERROR
Author	W.F.J. Mueller

ERRNAM

MESSAGE	<module>: Error in name string: '<string>'.
RECOVERY	Depending on the usage of the string the syntax must be checked.
SEVERITY	ER
Author	H.G.Essel

ILLKEYLEVEL

MESSAGE	'<Module>':The defined Keypad-Keys needs more dimensionlity than defined '<defined limit>' in U$TCSHK.
RECOVERY	Check the dimensionality of the defined Keypad-Keys and reduce them.
SEVERITY	ER
Author	Th. Kroll

ILLMODE

MESSAGE	The procedure has been called with an illegal mode of '<mode>'
RECOVERY	Signals, that the call of the procedure passed a mode parameter with an illegal value. Check what mode was intended and fix the bug in the CALLING program.
SEVERITY	ER
Author	Walter F.J. Mueller

INVTERMTYP

MESSAGE	Invalid terminal type
RECOVERY	The called programm runs not the specified terminal type. You have to use on other input/output device.
SEVERITY	ERROR
Author	W. Spreng

MISCERR

MESSAGE	Module <name> detected error <error message>
RECOVERY	Sorry, there is no specific recovery for this error. It depends on the error message.
SEVERITY	WARNING
Author	H.G.Essel

NOAUX

MESSAGE	No pointer for auxiliary keypad description structure defined
RECOVERY	Allocate this structure.
SEVERITY	ERROR

Author	W. Spreng

NOCONT

MESSAGE	No continuation line found for: '<line>'
RECOVERY	The source line requires a continuation line, but an end of file was detected
SEVERITY	ERROR
Author	H.G.Essel

NOOUTPUT

MESSAGE	<Module> tried to output <information>, but did not find a valid flag combination.
RECOVERY	Check the printflag mask passed to routine <module>
SEVERITY	WARNING
Author	T.Kroll

NOSTRDEL

MESSAGE	<module>: Missing string delimiter in string: <string>
RECOVERY	An input line has single string delimiters.
SEVERITY	E
Author	H.G.Essel

QUEFIRST

MESSAGE	was first element in queue
RECOVERY	No recovery
SEVERITY	INFORMATION

Author	W.F.J.Mueller

SMGNOID

MESSAGE	The SMG entity has not yet been created
RECOVERY	Signals, that a week request for a SMG ID was done and that this entity has not yet been created with a strong request.
SEVERITY	WA
Author	Walter F.J. Mueller

SUBSCRIPT

MESSAGE	Subscript error in <module> <explanation of error>
SEVERITY	WA
Author	W. Spreng

SUBSCRVIOL

MESSAGE	Subscript range check in SUBSTR(<string>,<begin>,<length>)
RECOVERY	Check the arguments of U$SUBST function.
SEVERITY	WARNING
Author	H.G.Essel

UTILERROR

MESSAGE	Can generate any message.
RECOVERY	Described in message text
SEVERITY	ER
Author	W. Spreng

WRHBOUND

MESSAGE	Array <array variable> has not suitable high bounds: <hbound(array variable)>, should be <allowed limit>.
RECOVERY	Correct the bounds of the referenced array. If found in U$CHOP: input string is truncated.
SEVERITY	ER
Author	K.Winkelmann

WRLBOUND

MESSAGE	Array <array variable> has not suitable low bounds: <lbound(array variable)>, should be <allowed limit>.
RECOVERY	Correct the bounds of the referenced array.
SEVERITY	ER
Author	K.Winkelmann

XABNOTFOUND

MESSAGE	<Module> tried to output <information>, but did not find a valid control pattern.
RECOVERY	The call of the procedure U$FINFO has had a wrong XAB argument. Correct it.
SEVERITY	ERROR
Author	M. Richter

GOOVME

Prefix	XVME_
Object name	GOOMSGLIB(XVME)
File name	XVME.MSG
Dataset	RZ88.XVME.MSG
Identifier	V01.00A
Facility	110

ACQNOTSTOP

MESSAGE	Acquisition stop error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

AEB_ACQ_ALR_STA

MESSAGE	Acquisition aleady started
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_ACQ_ALR_STO

MESSAGE	Acquisition aleady stopped
RECOVERY	

SEVERITY	ERROR
Author	U.Post

AEB_CLR_MOD_ERR

MESSAGE	Module could not be cleared.
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_ERROR

MESSAGE	Some severe errors occured: <description>
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_INIT_FAILED

MESSAGE	Initialization failure
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_INV_CMD

MESSAGE	Invalid command
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_INV_MOD_ID

MESSAGE	Invalid module ID.
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_INV_SUBCR

MESSAGE	Wrong subcrate specification.
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_INV_TT

MESSAGE	Invalid TT
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_LRS18XX_N

MESSAGE	LRS18XX not in n mode
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_MOD_NOTF

MESSAGE	Module not found

RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_MOD_UNEXP

MESSAGE	Unexpected module found
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_NO_LIST

MESSAGE	No readout list is loaded
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_READ_ERROR

MESSAGE	Readout error
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_SEV_FB_ERR

MESSAGE	Severe FASTBUS error
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_SUCCESS

MESSAGE	Successful completion
RECOVERY	
SEVERITY	SUCCESS
Author	H.G.Essel

AEB_SYNC_ERROR

MESSAGE	Readout sync error
RECOVERY	
SEVERITY	ERROR
Author	U.Post

AEB_TRIG_ERROR

MESSAGE	Trigger number out of range
RECOVERY	Trigger number must be in the range 0-15.
SEVERITY	ERROR
Author	W. Ahner

AEB_WARNING

MESSAGE	Warning, some errors occured: <description>
RECOVERY	
SEVERITY	WARNING
Author	H.G.Essel

AEB_WARNING

MESSAGE	Warning, some errors occured: <description>

RECOVERY	
SEVERITY	WARNING
Author	W. Ahner

BUFWAITTIME

MESSAGE	FEP timeout error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

BUSERROR

MESSAGE	Bus Error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

CMDNOTINST

MESSAGE	Command not installed
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

CPUINHALT

MESSAGE	CPU halt error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

CREATEXERR

MESSAGE	Create Exchange Error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

CVI_ACQ_ALR_STA

MESSAGE	Acquisition already started
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_ACQ_ALR_STO

MESSAGE	Acquisition already stopped
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_CAM_BUS_ERR

MESSAGE	CAMAC bus error
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_ERROR

MESSAGE	Some severe errors occured: <description>

RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_INV_CMD

MESSAGE	Invalid command
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_INV_TT

MESSAGE	Invalid transfer type
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_LIST_LD_ERR

MESSAGE	Error loading readout/init/reset list
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_NO_RD_LIST

MESSAGE	No readout list loaded
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_READ_ERROR

MESSAGE	Readout Error
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_SUCCESS

MESSAGE	Successful completion
RECOVERY	
SEVERITY	SUCCESS
Author	W. Ahner

CVI_TRIG_ERROR

MESSAGE	Trigger type out of range
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

CVI_US_PR_ERR

MESSAGE	Error in user program
RECOVERY	
SEVERITY	ERROR
Author	W. Ahner

DMABUSERROR

MESSAGE	DMA Bus Error

RECOVERY	
SEVERITY	ERROR
Author	W.Ott

EBLASTBUF

MESSAGE	Event builder sent last buffer.
RECOVERY	After STOP ACQUISITION the event builder sends a buffer with current_1 = -1.
SEVERITY	ERROR
Author	H.G.Essel

EBREADTIMOUT

MESSAGE	Event builder read timeout.
RECOVERY	One or more FEP's did not announce a subevent to the event builder. A complete event could not be built. Check trigger logic. Stop the acquisition and start again.
SEVERITY	ERROR
Author	H.G.Essel

ERROR

MESSAGE	VME Error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

ESONEERR

MESSAGE	ESONE error
RECOVERY	

SEVERITY	ERROR
Author	W.Ott

FEPBUFWAIT

MESSAGE	Wait buffer error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

ILLEGALINSTR

MESSAGE	Illegal Instruction
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

INVCRATE

MESSAGE	Invalid crate
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

INVPROCID

MESSAGE	Invalid processor ID
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

MISSFEP

MESSAGE	FEP missing error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

MSG

MESSAGE	Message from crate c, offset o, subcrate s, control c.
RECOVERY	Specific error message follows.
SEVERITY	ERROR
Author	H.G.Essel

NOCMD

MESSAGE	no command
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

NOFREEBUFFER

MESSAGE	No free buffer
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

NOMSG

MESSAGE	no message

RECOVERY	
SEVERITY	ERROR
Author	W.Ott

NOREADLIST

MESSAGE	no readoutlist found
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

SEMISMATCH

MESSAGE	Subevent mismatch, ID=#
RECOVERY	There is a severe error with the trigger module. Stop data acquisition and check trigger module. The internal subevent counters are out of order. They can be inspected by command SET VME TRIGGER
SEVERITY	ERROR
Author	H.G.Essel

SENDXERR

MESSAGE	Send message error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

SKIPEVT

MESSAGE	Invalid Subevent
RECOVERY	
SEVERITY	ERROR

Author	W.Ott

SMGINVATTR

MESSAGE	Invalid window attribute
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

SMGINVCOL

MESSAGE	Invalid cursor column
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

SMGINVROW

MESSAGE	Invalid cursor row
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

SMGINVWIN

MESSAGE	Invalid window number
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

SUCCESS

MESSAGE	Successful completion

RECOVERY	
SEVERITY	SUCCESS
Author	W.Ott

TEXT

MESSAGE	Additional information: text
RECOVERY	Depends on error
SEVERITY	ERROR
Author	H.G.Essel

TIMEOUT

MESSAGE	Timeout
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

TOOMUCHKEYS

MESSAGE	Too much keys in menue
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

TRANSERROR

MESSAGE	Transfer error
RECOVERY	
SEVERITY	ERROR

Author	W.Ott

TRANSHOSTIMEOUT

MESSAGE	Host timeout error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

TRANSVMETIMEOUT

MESSAGE	VME timeout error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

UNVCNAF

MESSAGE	INCALID CNAF
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

VSBBUSERR

MESSAGE	VSB Bus Error
RECOVERY	
SEVERITY	ERROR
Author	W.Ott

ZERODIVIDE

MESSAGE	Divide by Zero Error

RECOVERY	
SEVERITY	ERROR
Author	W.Ott

GOOSY Glossary

Analysis Manager ($ANL) Part of the analysis program controlling the data I/O and the event loop.

$ANL The Analysis program as a GOOSY component. Runs in a subprocess named GN_env___$ANL.

$DBM The Data Base Manager as a GOOSY component. Runs in a subprocess named GN_env___$DBM.

$DSP The Display Program as a GOOSY component. Runs in a subprocess named GN_env___$DSP.

$TMR The Transport Manager as a GOOSY component. Runs in a subprocess named GN_env___$TMR.

ATTACH Data Bases, Pools, and Dynamic Lists must be attached before they can be used. The ATTACH operation specifies the protection mode for Data Base Pools.

Branch The CAMAC parallel branch connects up to seven CAMAC crates to a computer Interface, e.g. to the MBD.

Buffer GOOSY buffers have a standard buffer header describing the content of the buffer through type/subtype numbers. A GOOSY buffer may contain list mode data (events) file headers, or other kind of data. Buffers can be sent over DECnet and copied from/to tape and disks. Most GOOSY buffers contain buffer Data Elements.

Buffer Data Element A data structure preceeded by a 4 word header stored in a buffer. The header keeps information about the size and the type of the buffer Data Element.

Buffer Unpack Routine A buffer unpack routine copies one event from the buffer into an event Data Element. It has to control the position of the events in the buffer. It gets passed the pointer to the buffer as argument.

CAMAC **C**omputer **A**utomated **M**easurement **a**nd **C**ontrol. A standard for high-energy physics and nuclear physics data acqusition systems, defined by the ESONE (**E**uropean **S**tandard **O**n **N**uclear **E**lectronics) committee between 1966 and 1969.

CONDITION In contrast to SATAN, GOOSY conditions are independent of spectra. Besides the multi window conditions which are similar to SATAN analyzer conditions, GOOSY provides window-, pattern-, composed- and userfunction-conditions. Each condition has counters associated for true/false statistics. Conditions can be executed in a Dynamic List or by macro the $COND in an analysis routine. Each condition can be used as filter for spectrum accumulation or scatter plots.

CONNECT A calibration can be connected to any number of spectra with the GOOSY command `CALIBRATE SPECTRUM`.

CVC CAMAC VSB Computer. A CAMAC board with a 68030 processor running Lynx, OS9 or pSOS. It can be equipped with ethernet and SCSI and VSB.

Data Base A Data Base is located in a file and has a Data Base name. It is recommended to use the same name for the file and the Data Base. The file type should be .SEC. A logical name may be defined for the Data Base name. To activate a Data Base it must be mounted. It is dismounted during a system shutdown or by command. If a Data Base runs out of space, it can presently NOT be expanded.

Data Base Directory Similar to a VMS disk, GOOSY Data Bases are organized in Directories. They must be created.

Data Base Manager ($DBM) This is a program executing all commands to handle Data Bases. It may run directly in DCL or in a GOOSY environment.

Data Base Pool The storage region of a Data Base is splitted in Pools. All Data Elements are stored in Pools. A Pool can be accessed by a program with READ ONLY protection or with READ/WRITE protection. Pools must be created. They are automatically expanded if necessary, up to the space available in a Data Base.

Data Element A Data Element is allocated in a Data Base Pool. Its name is kept in a Directory. Data Elements can be of atomic Types (scalars or arrays), or of the structure Type (PL/1 structures). Besides the data structure a Data Element can be indexed (one or two dimensional). Such Data Elements are called name arrays. Each name array member has its own data and Directory entry.

Data Element Member Similar to PL/1, the variables in a structure are called members.

Data Element Type GOOSY Data Elements can be PL/1 structures. The structure declarations must be in a file or text library module. They are used to create a Data Element Type in the Data Base and can be included in a program to access the Data Element.

Dynamic List A Dynamic List has several Entries, each specifying an action like condition check or spectrum accumulation. It is executed for each event in the analysis program. The Entries are added or removed by commands even without stopping the analysis.

Dynamic List Entry An Entry in a Dynamic List keeps all information to execute an action. For example, an accumulation Entry contains the spectrum name, an object and optional a condition and an increment parameter.

Dynamic List Executor The part of the analysis program which scans through a Dynamic List for each event executing the actions specified by the Entries.

Environment The Transport Manager and the analysis programs run only in a GOOSY environment which has to be created first. They are started by specific commands. The Display and the Data Base Manager may run under DCL or in a GOOSY environment. The display must run in a GOOSY environment if scatter plots are used. The main difference is that in an environment several programs are 'stand by', whereas in DCL you can run only one program at a time.

Event Packet of data in the input or output stream which is processed by the same program part (see event loop).

Event Buffer Data Element A data structure preceeded by a 4 word header stored in a buffer. The header keeps information about the size and the type of the event buffer Data Element. The event buffer Data Element is copied by unpack routines to event Data Elements.

Event Data Element A Data Element in a Data Base which is used to store events. Event Data Elements are used to copy events from an input buffer into the Data Base or from the Data Base into an output buffer.

Event Unpack Routine An event unpack routine copies one event from the buffer into an event Data Element. Different from a buffer unpack routine, it gets passed the pointer to the event in the buffer as argument.

GOOSY Components GOOSY is composed of components, i.e. programs like the Transport Manager $TMR, the Analysis Program $ANL, the Display $DSP and the Data Base Manager $DBM. Data Base Manager and Display program may be envoced under DCL in a 'stand alone' mode. $TMR and $ANL can run only in a GOOSY environment. Components run in an environment as VAX/VMS subprocesses of the terminal process.

GOOSY Prompter If GOOSY components run in an environment, their commands are the input to the GOOSY prompter. The GOOSY prompter is entered by `GOOSY` and prompts with `SUC: GOOSY>`. Now you can enter GOOSY commands which are dispatched to the appropriate GOOSY components for execution. Single GOOSY commands can be executed from DCL preceding them by `GOOSY`. The prompter exits after the command termination. **The GOOSY prompter can only be used after an environment was created!**

J11 This is an auxiliary crate controller based on a PDP 11/73 processor (type CES 2180 Starburst). Has full PDP instruction set including floating point arithmetic. A J11 running under RSX/11S controls one CAMAC crate and sends the data via DECnet to a VAX.

LAM **L**ook **A**t **M**e. A signal on the CAMAC Dataway, which may request a readout (CAMAC interrupt).

LOCATE In a program, any Data Element must be located, before it can be used. The LOCATE operation returns the pointer to the Data Element. The macro $LOC provides a convenient way to locate spectra, conditions or arbitrary Data Elements.

Mailbox An interprocess communication method provided by VMS. Processes on the same node can send/receive data through mailboxes.

MBD Microprogrammed **B**ranch **D**river from BiRa Systems Inc. supports the protocol of the CAMAC parallel *Branch*, defined by the *CAMAC* standard (GOLDA equivalent: CA11-C). This is an interface between CAMAC and a VAX. It gets data from the crate controllers (J11) and sends them to the transport manager running on a VAX.

MOUNT A GOOSY Data Base must be mounted before it can be accessed. The `MOUNT` operation connects the Data Base name with the Data Base file name.

Object To increment a spectrum or execute a condition, the Dynamic List executor needs a value for the spectrum channel, or a value to compare to window limits. These values are called objects. An object must be a member of a Data Element.

Picture A Picture is a complex display. A picture is a set of up to 64 frames with spectra and/or scatterplots. Once created and specified they remain in a Data Base independent of programs. They are displayed by `DISPLAY PICTURE` command. Pictures are composed of frames.

Picture Frame Each frame is a coordinate system for a spectrum or scatter plot. Up to 64 different frames may inserted to a picture.

Prompter Command interface for GOOSY environment. The GOOSY prompter is called by DCL command `GOOSY`. Then all commands are delivered to the environment components for execution.

Scatter Plot The GOOSY display component can display any pairs of Data Element members event by event in scatter plot mode (live mode). Several scatter plots can be displayed on one screen (pictures). Scatter plots are executed in Dynamic Lists and may be filtered by conditions.

Spectrum A GOOSY spectrum differs from a SATAN analyzer in that there are no windows or conditions associated. A spectrum can be filled in a Dynamic List Entry or in an analysis routine by macro $ACCU.

STARBURST This is an auxiliary crate controller based on a PDP 11/73 processor (type CES 2180 Starburst). Has full PDP instruction set including floating point arithmetic. Each CAMAC crate is controlled by one STARBURST running a standalone program. The STARBURST reads out the crate and sends the data to the MBD.

Supervisor Each environment has a supervisor component. The supervisor dispatches messages between the GOOSY prompter and the environment components.

Transport Manager ($TMR) This program acts as data buffer dispatcher. It gets data buffers from the CAMAC branch (MBD) or via DECnet from a single CAMAC crate (J11) or from a disk/tape file and writes them to disk/tape files, DECnet, and mailboxes. It executes all CAMAC control commands. The $TMR runs only in a GOOSY environment.

Unpack Routine An unpack routine copies one event from the buffer into an event Data Element. There are two types: buffer and event unpack routines. Buffer unpack routines control the whole buffer, event unpack routines only one event.

Index

Contents