GOOSY
Id.: ANAL
Version: 1.0
Date: 14–Jun–1988
Revised: May, 20 1992

GSI Online Offline SYstem

GOOSY Data Acquisition and Analysis

H.G.Essel

May, 20 1992

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

Fn, PFn, 1, Do, or 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.

GOLD, <GOLD>— The PF1 key is called 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 PFn of the rightmost VTx20 compatible keypad are named Fn and the 20 Fn keys on the top of each VTx20 compatible keyboard are named PFn on a PERICOM.

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

Enter — If your terminal is connected to IBM, the Enter key terminates all command lines.

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

- `Ctrl` `Z` means hold down the `Ctrl` key and type the letter `Z`.

`PFn` **key** — The `PFn` followed by a number means that you must press the `PFn` key and *then* type the number. Here is an example:

- `PF1` `6` press the `PF1` key and then type the number 6 on the main keyboard.

`PFn` **or** `Fn` **keys** — Any `PFn` or `Fn` key means that you just press this key. Here is an example:

- `PF2` means press the `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.

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

1.3 Intended Audience

This documentation describes the data acquisition and analysis. It assumes that the reader is familiar with basic VAX-VMS concepts and commands. For VAX beginners the 'VMS Introduction' is recommended. For GOOSY beginners the 'GOOSY Introduction' is recommended.

1.4 Overview

- Section 2:
  A short description of GOOSY environments and their components.
- Section 3:
  Transport Manager structure. The TMR controls the frontend equipment and dispatches the data.
- Section 4:
  Analysis Manager structure. The AMR controls the analysis.
- Section 5:
  Analysis program structure. Analysis routines. Dynamic analysis.

Chapter 2

GOOSY Environment

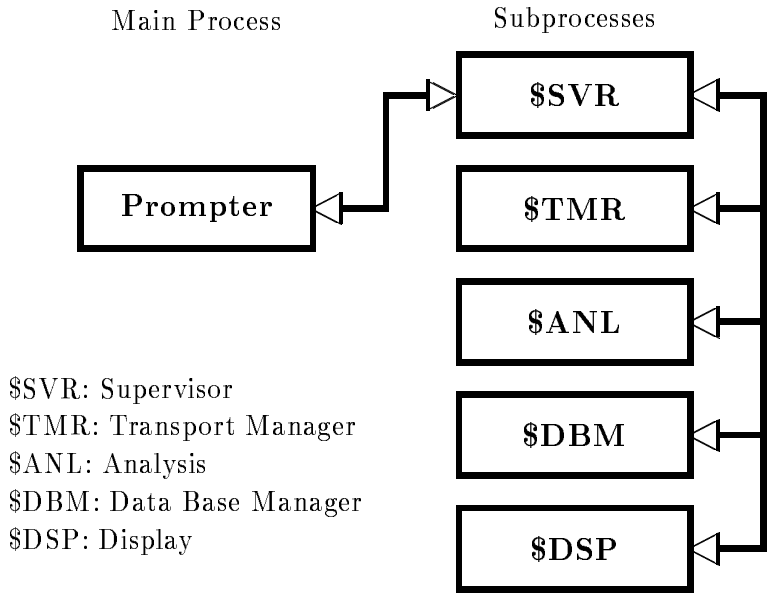


Figure 2.1: GOOSY components shown as VMS processes.

2.1 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. 2.1. 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 2.1.3 to create a GOOSY environment.

2.1.1 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 2.2 on page 12 shows the keypad layout.

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

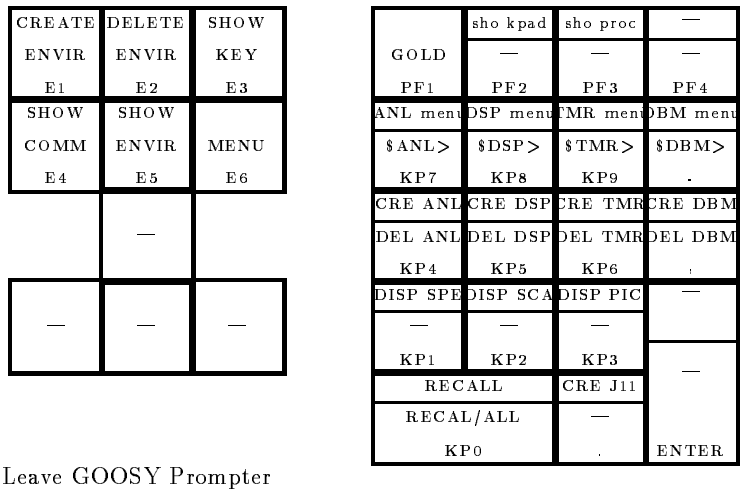
```
   <KP_7>     ! $ANL : Analysis commands menu
   <KP_8>     ! $DSP : Display commands menu
   <KP_9>     ! $TMR : Transport Manager commands menu
   <KP_MINUS> ! $DBM : Data Base Manager commands menu
```

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 2.2: The Special Keypad Layout for the GOOSY prompter.

2.1.2 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> 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 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> 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 2.1 on page 10 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.

2.1.3 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`.

2.1.4 Deletion of Environments

The DCL command

```
$ DLENVIR
```

deletes the present environment including all components (subprocesses).

2.1.5 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 3

GOOSY Transport Manager

3.1 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

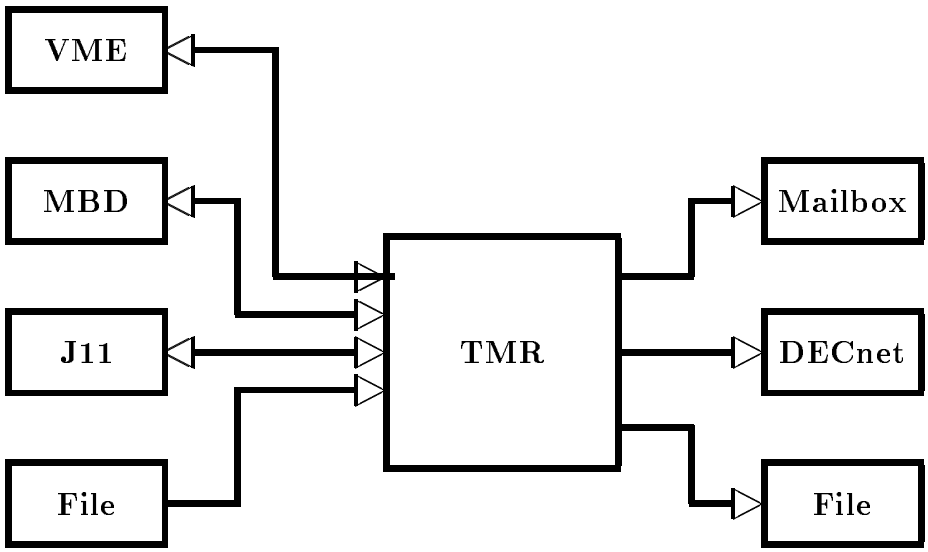


Figure 3.1: The input and output channels of the Transport Manager

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.

3.2 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.

3.3 Input Channels

3.3.1 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.

3.3.2 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_SUS
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
```

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.3.6 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 3.6 for that. These routines must be linked into a sharable image by the DCL command `LSHARIM` (see section 4.5.1 page 52.

3.4 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
```

3.5 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.

3.5.1 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.

3.5.2 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.

3.5.3 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 37). 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
```

3.6 Loading Private Routines

After the startup of the Transport Manager one can load private routines to be called by the `START-STOP ACQUISITION` commands.

3.6.1 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 4.5.1 on page 52). 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.

3.6.2 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
```

3.7 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.

3.8 Miscellaneous Commands

Besides the commands described above the following commands are available.

3.8.1 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.

3.8.2 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.

3.8.3 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
```

3.8.4 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.

3.9 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.

3.9.1 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
```

3.9.2 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.

3.9.3 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
```

3.9.4 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
```

3.9.5 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 4

GOOSY Analysis Manager

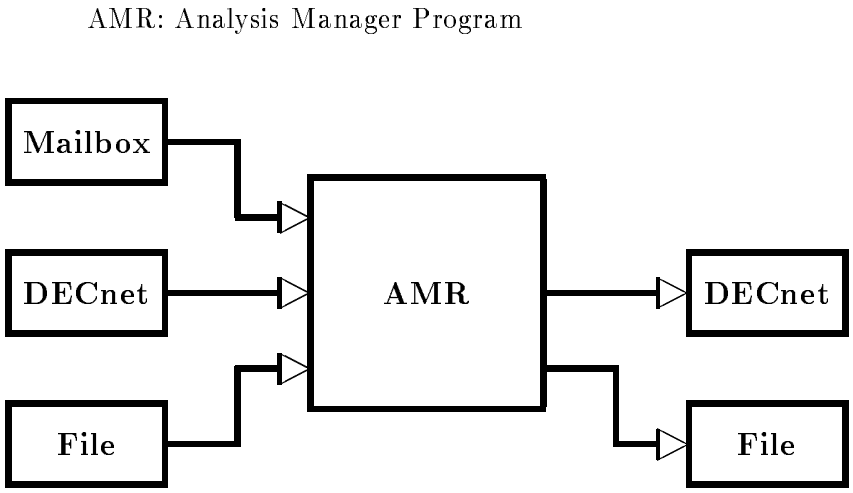


Figure 4.1: The input and output channels of the Analysis Manager (analysis program)

4.1 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.

4.2 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.

4.2.1 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.

4.3 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.

4.3.1 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).

4.3.2 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.

4.3.3 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.

4.4 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.

4.4.1 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.

4.4.2 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.

4.5 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.

4.5.1 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.

4.5.2 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.

4.5.3 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 ...
```

4.6 Re-Initialize Analysis

4.6.1 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.

4.6.2 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.

4.7 Miscellaneous Commands

4.7.1 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
```

4.7.2 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
```

4.7.3 Synchronizing the Output

The analysis output can be synchronized by

```
GOOSY> SET ANALYSIS /SYNCHRON
GOOSY> SET ANALYSIS /NOSYNCHRON
```

4.7.4 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
```

4.7.5 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 5

GOOSY Analysis

5.1 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.

5.1.1 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 5.1 on page 61 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.

5.1.2 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.

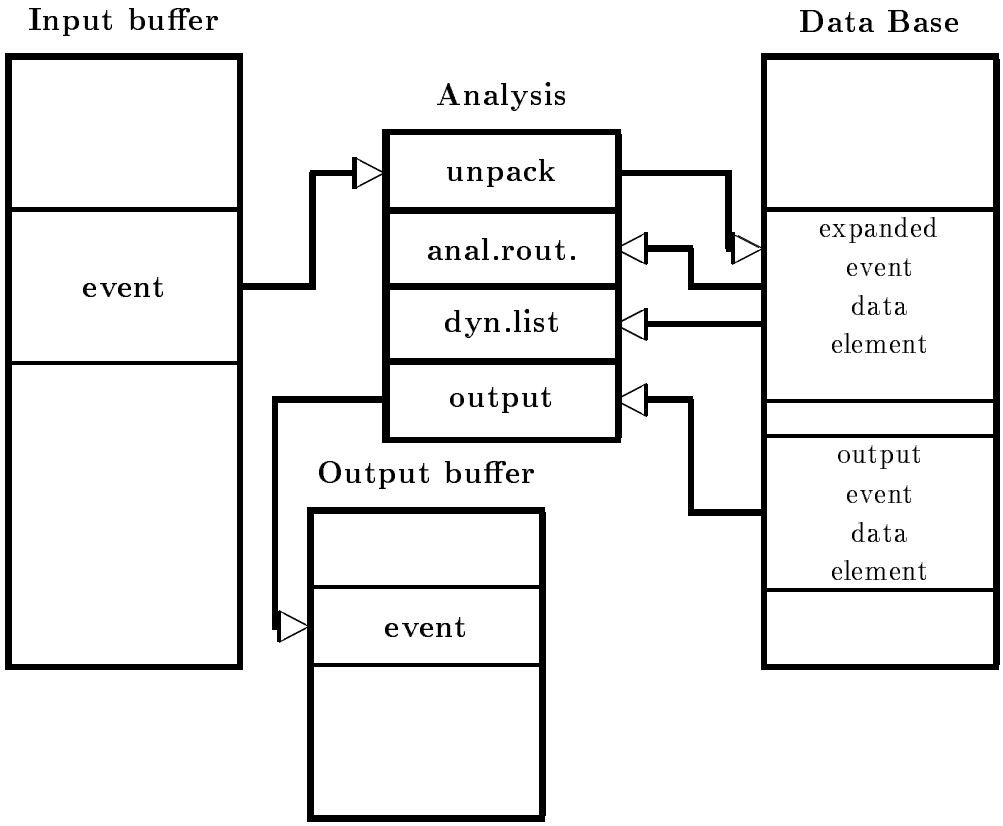


Figure 5.1: Moving events between I/O buffers and data bases in the event loop.

5.1.3 Getting the Data

As shown in figure 5.1 on page 61 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.

5.1.4 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.

5.2 ONLINE Analysis Design

5.2.1 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.

5.2.2 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.

5.3 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 52.

The user can decide which method is most suited for his purpose. In all cases the routines must follow the some rules.

5.3.1 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.

5.3.2 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.

5.4 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.

5.4.1 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.

5.4.2 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.

5.5 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.

5.6 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.

5.6.1 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.

5.6.2 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.

5.7 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 52).

5.8 Analysis Macros

The macro calls for analysis routines can be inserted by the LSEDIT editor using the F_8 key.

5.8.1 $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       */
```

5.8.2 $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 */
```

5.8.3 $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 */
```

5.8.4 $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 */
```

5.8.5 $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 */
```

5.9 Dynamic Analysis

5.9.1 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
```

5.9.2 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
```

5.9.3 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.

5.9.4 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.

5.9.5 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.

5.10 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
```

Appendix A

GOOSY Commands

$ CLOSE ETHERNET

$ CLOSE ETHERNET /SHOW

PURPOSE	Close Ethernet link.
PARAMETERS	
/SHOW	Show statistics.
EXAMPLE	$ CL ETH

Description

FUNCTION	-
File name	I$ACVME.PPL
Action rout.	I$ACVME_CLO_ETH
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	20-Jul-1989

$ SET GNA ETHERNET

$ SET GNA ETHERNET Link Read Write Acknow
Wretry Lretry output
/[NO]SWAP
/[NO]ISTREAM
/[NO]OSTREAM
/[NO]DEBUG

PURPOSE SET Ethernet modes.

PARAMETERS

Link Character def="0 :30:10" Delta time for link check, i.e. 3 sec = '0 ::3'

Read Character def="0 :00:30" Delta time for read timeout, i.e. 3 sec = '0 ::3' This time should be larger than write timeout times number of write retries times maximum number of physical buffers per logical buffer.

Write Character def="0 :00:05" Delta time for write timeout, i.e. 3 sec = '0 ::3'

acknow Integer Number of acknowledges to sent back for each buffer.

Wretry Integer Number of write retries.

Lretry Integer Number of ignored link checks.

Output Character Optional output file.

/SWAP Enable swapping. Def=/SWAP

/ISTREAM Enable stream input. Def=/NOISTREAM

/OSTREAM Enable stream output. Def=/NOOSTREAM

/DEBUG Enable debug output. Def=/NODEBUG

EXAMPLE $ SET GNA ETH READ="0 :00:50" /ISTR

Description

FUNCTION	-
File name	I$ACVME.PPL
Action rout.	I$ACVME_SET_GNA
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	20-Jul-1989

$ SHOW GNA ETHERNET

$ SHOW GNA ETHERNET /FULL /CLEAR

PURPOSE	Show Ethernet information.
PARAMETERS	
/FULL	Show full information.
/CLEAR	Clear counter.
EXAMPLE	$ SHO GNA ETH /F

Description

FUNCTION	-
File name	I$ACVME.PPL
Action rout.	I$ACVME_SHO_GNA
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	20-Jul-1989

ATTACH ANALYSIS

ATTACH ANALYSIS

PURPOSE	Reinitialize analysis after DETACH ANALYSIS
NOTE	Dynamic lists must be attached again.

Description

FUNCTION	This command calls $IBUFFER to reinitialyze the analysis. The data base is detached and attached.
File name	I$ANACM.PPL
Action rout.	I$ANACM_ATT
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

ATTACH DYNAMIC LIST

ATTACH DYNAMIC LIST dyn_list dyn_dir base node
/FAST

PURPOSE Attach dynamic list

PARAMETERS

dyn_list Dynamic list name specification
required common default

dyn_dir Default directory
common default:'$DYNAMIC'

base Default data base name
common default:'DB'

node Default node name
common default:'E'

/FAST No execution or freeze bits are checked. Execution bit is set, however. No counters are incremented. Only the following data types are supported:
BIN FLOAT(24) for window, spectra objects
BIT(32) ALIGNED for patterns.
Spectrum type must be R, increment is always 1.
Master entries, scatter plots and user functions are not affected.

EXAMPLE -

Caller M$DLCMD

Author H.G. Essel

File name M$AATDL.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$AATDL(CV_DYN_LIST,CV_DYN_DIR,
CV_BASE,CV_NODE,I_FAST)

ARGUMENTS

CV_DYN_LIST Dynamic list name specification

CV_DYN_DIR Default Directory

CV_BASE Default Data Base name

CV_NODE Default node name

I_FAST Fast execution
IF 1, no execution or freeze bits are checked.
No counters are incremented. Only the following data types are supported:
BIN FLOAT(24) for window, spectra objects
BIT(32) ALIGNED for patterns.
Spectrum type must be R, increment is always 1.

FUNCTION Attach dynamic list

REMARKS Module is an action routine.

EXAMPLE -

CALCULATE FASTBUS PEDESTAL

CALCULATE FASTBUS PEDESTAL loop throff thrfact
pedoff pedfact sample trigger
VMEcrate,processor ID dummy crate node
/ON/OFF [=ONOFF]
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Set fastbus pedestal subtraction on/off.

PARAMETERS

loop integer (def=100)
Number of events to measure.

throff integer (def=0)
Threshold offset [channels]to add to mesured value

thrfact real (def=1.)
Factor for mesured threshold.

pedoff integer (def=100)
Pedestal offset [channels]to add to mesured value.

pedfact real (def=1.)
Factor for mesured pedestal.

sample integer (def=100)
Sample interval [events]. After "sample" events one event will be not compressed.

trigger integer (def=1)
Trigger number

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy	NOT used
crate	Crate number
node	NET node
/ON/OFF	Switch compression ON or OFF
/ALL/FEP/EB	Select processor
/CVI/CAV/EBI	Select processor by controller
/[NO] LOAD	Do [NOT]execute. Default= /LOAD

EXAMPLE SET VME TRIG

Description

FUNCTION	Measure and calculate pedestals and thresholds.
File name	I$ACV_CALC_PED.PPL
Action rout.	I$ACV_CALC_PED
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

CAMAC CLEAR

CAMAC CLEAR C=c

PURPOSE	Generate Dataway clear
PARAMETERS	
C=c	Number of the crate, in which the dataway clear has to be performed.
EXAMPLE	CAMAC CLEAR 1
Action rout.	I$MCCCC
Author	Walter F.J. Mueller

Remarks

File name	I$MCCCC.PPL
Dataset	-
REMARKS	-

Description

CALLING	@CALL I$MCCCC(I_C);
ARGUMENTS	
I_C	BIN FIXED(15) [INPUT] Crate number between 1 and 7 in which the dataway clear has to be executed.
FUNCTION	This procedure calls I$CCCC to perform a dataway clear (C) in the specified crate.
REMARKS	-
EXAMPLE	@CALL I$MCCCC(1);

CAMAC CNAF

CAMAC CNAF C=c N=n A=a F=f DATA=d Branch=b

PURPOSE Perform a single CAMAC action

PARAMETERS

C=c Crate number, between 1 and 7.
Replaceable default = 1

N=n Station number, between 1 and 31.
Replaceable default = 1

A=a Subaddress, between 0 and 15.
Replaceable default = 0

F=f Function code, between 0 and 31.
Replaceable default = 0

DATA=d Data word. Should be specified if the function code is between 16 and 23 and will be ignored otherwise.
NOTE: The data word may be entered also in binary (e.g. %B101), octal (e.g. %O123), or hexadezimal (e.g. %XA1B) if convienient.
Replaceable default = 0

Branch=b Branch number. Used for the translation of
CAMAC_BRANCH_b

EXAMPLE CAMAC CNAF C=1 N=12 A=3 F=16 DATA=%O777

Action rout. I$MCCNA

Author Walter F.J. Mueller

Remarks

File name I$MCCNA.PPL

REMARKS -

Description

CALLING @CALL I$MCCNA(I_C,I_N,I_A,I_F,L_DATA);

ARGUMENTS

I_C BIN FIXED(15) [INPUT]
Crate number, between 1 and 7.

I_N BIN FIXED(15) [INPUT]
Station number, between 1 and 31.

I_A BIN FIXED(15) [INPUT]
Subaddress, between 0 and 15.

I_F BIN FIXED(15) [INPUT]
function code, between 0 and 31.

L_DATA BIN FIXED(31) [INPUT]
Data word to be written if function is between 16 and 23, ignored otherwise.

I_B BIN FIXED(15) [INPUT]
Branch number. This number is used for the translation of CAMAC_BRANCH_b.

FUNCTION This procedure calls I$CFSA to execute the given CAMAC action and displays the result.

REMARKS -

EXAMPLE @CALL I$MCCNA(1,1,0,16,1);

CAMAC DEMAND

CAMAC DEMAND C=c
/ENABLE/DISABLE/TEST

PURPOSE	Enable, disable or test Crate Demand
PARAMETERS	
C=c	Number of the Crate, for which the demand has to be enabled, disabled or tested.
/ENABLE	The demand will be enabled and the status displayed.
/DISABLE	The demand will be disabled and the status displayed
/TEST	Only the status will be displayed, this is the default. It is signaled, whether demands are present in this crate.
EXAMPLE	CAMAC DEMAND 1
Action rout.	I$MCCCD
Author	Walter F.J. Mueller

Remarks

File name	I$MCCCD.PPL
Dataset	-
REMARKS	The /ENABLE and /DISABLE functions may interfere with other running MBD programs, be carefull !!!

Description

CALLING	@CALL I$MCCCD(I_C,C_MODE);
ARGUMENTS	

I_C BIN FIXED(15) [INPUT]
Number of the crate, in which the demand has to be enabled, disabled or tested.

C_MODE CHAR(*) VAR [INPUT]
Mode qualifier, may have the values:

/ENABLE	Demand will be enabled
/DISABLE	Demand will be disabled
/TEST	Demand enablement and the presents of demands will be tested.

FUNCTION If C_MODE is '/ENABLE' or '/DISABLE', the demand in the specified crate is enabled or disabled by a call of I$CCCD. Afterwards or if C_MODE is '/TEST' the status of the demand enablement and the presents of demands is tested with a call of I$CTCD and I$CTGL and displayed.

REMARKS -

EXAMPLE @CALL I$MCCCD(1,'/TEST');

CAMAC INHIBIT

CAMAC INHIBIT C=c
/SET/CLEAR/TEST

PURPOSE	Set, clear or test Dataway inhibit
PARAMETERS	
C=c	Number of the crate, in which the inhibit has to be set, cleared or tested.
/SET	The inhibit will be set and the status displayed.
/CLEAR	The inhibit will be cleared and the status displayed
/TEST	The status of the inhibit will only be displayed.
EXAMPLE	CAMAC INHI 1 /CLEAR
Action rout.	I$MCCCI
Author	Walter F.J. Mueller

Remarks

File name	I$MCCCI.PPL
Dataset	-
REMARKS	-

Description

CALLING	@CALL I$MCCCI(I_C,C_MODE);
ARGUMENTS	
I_C	BIN FIXED(15) [INPUT] Number of crate, for which the Inhibit has to be set, cleared or tested.

C_MODE CHAR(*) VAR [INPUT]
Mode qualifier, may have the values:

/SET	Will set inhibit
/CLEAR	Will clear inhibit
/TEST	Will test and display inhibit status

FUNCTION This procedure calls I$CCCI to set or clear the inhibit in the selected crate and I$CTCI to test the inhibit status.

REMARKS -

EXAMPLE
```
@CALL I$MCCCI(1,'/TEST');
```

CAMAC INITIALIZE

CAMAC INITIALIZE C=c

PURPOSE	Generate Dataway Initialize
PARAMETERS	
C=c	Number of crate, in which the dataway initialize is to be executed.
EXAMPLE	CAMAC INIT 1
Action rout.	I$MCCCZ
Author	Walter F.J. Mueller

Remarks

File name	I$MCCCZ.PPL
Dataset	-
REMARKS	-

Description

CALLING	@CALL I$MCCCZ(I_C);
ARGUMENTS	
I_C	BIN FIXED(15) [INPUT] Crate number between 1 and 7 in which the dataway initialize has to be performed.
FUNCTION	This procedure calls I$CCCZ to do a dataway init in the specified crate.
REMARKS	-
EXAMPLE	@CALL I$MCCCZ(1);

CAMAC SCAN

CAMAC SCAN C=c N=n F=f /CRATE/STATION/ADDRESS

PURPOSE Perform a crate scan

PARAMETERS

C=c Number of the Crate to be inspected.

N=n Number of the Station to be inspected for a station or address scan.

F=f Function code to be used for an address scan.

/CRATE Signals crate scan. Will try in the specified crate all station - subaddress combinations for function code F=0 and reports for each station:
the highest subaddress with an Q or X response
the 'strongest' response (X or Q) found

/STATION Signals station scan, this is the default. Will try in the specified crate and station all subaddress - function combinations and reports the Q and X response for each combination:
- indicates no X, no Q
X indicates X but no Q
q indicates Q but no X (strange combination !!!)
Q indicates X and Q

/ADDRESS Signals address scan. Will read out in the specified crate all subaddresses of either all or the specified station with the given function code and reports the value if there was a Q response.

Action rout. I$MCSCN

Author Walter F.J. Mueller

Remarks

File name I$MCSCN.PPL

Dataset -

REMARKS -

EXAMPLE -

Description

CALLING @CALL I$MCSCN(I_C,I_N,I_F,C_MODE,B_DP);

ARGUMENTS

I_C BIN FIXED(15) [INPUT]
Crate to be scanned, must be between 1 and 7.

I_N BIN FIXED(15) [INPUT]
Station to be scanned for a station scan, must be between 1 and 31.

I_F BIN FIXED(15) [INPUT]
Function code to be used for a address scan, must be between 0 and 31.

C_MODE CHAR(*) VAR [INPUT]
Scan mode, valid are:

/CRATE	Perform crate scan, will use I_C.
/STATION	Perform station scan, will use I_C, I_N.
/ADDRESS	Perform address scan, will use I_C, I_F.

B_DP BIT(*) ALIGNED [INPUT]
Default pattern, is used to determine whether I_N was specified in an address scan or not.

FUNCTION For details look in the command description.

REMARKS -

EXAMPLE @CALL I$MCSCN(1,1,0,'/CRATE');

CLOSE FILE

CLOSE FILE

PURPOSE	Close data input file.
EXAMPLE	CLOSE FILE
NOTE	The acquisition must be initialized with /FILE

Description

FUNCTION	Closes a file for data input. The input is stopped by STOP ACQUIS.
File name	I$ACQ_CLO_FIL.PPL
Action rout.	I$ACQ_CLO_FIL
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	20-AUG-1987

CLOSE OUTPUT FILE

CLOSE OUTPUT FILE

PURPOSE	Close list mode dump file
PARAMETERS	
EXAMPLE	CLOSE O F
NOTE	This command is called by STOP OUT FILE which should be used. Actually the output must be stopped anyway.

Description

FUNCTION	Close list mode file.
File name	I$ACQ_CLO_LMD.PPL
Action rout.	I$ACQ_CLO_LMD
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

CNAF VME

CNAF VME VMEcrate,processor C N A F times data ID
dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Execute CNAF

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

C Crate number

N Station

A Subaddress

F Function

times Repetition

data Data

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE CNAF VME 1,1 1 1 2 24

Description

FUNCTION	Execute CNAF.
File name	I$ACV_CNAF_VME.PPL
Action rout.	I$ACV_CNAF_VME
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

COPY FILE

COPY FILE file outfile skip buffers

PURPOSE Output GOOSY list mode data file (called in MUTIL).

PARAMETERS

file required string replace
List mode data file.

output string replace
Required output file.

skip integer default=0
Optional number of buffers to skip.

buffers integer default=100000000
Optional number of buffers to output

Caller MUTIL

Author H.G.Essel

Example

```
$ MUTIL COPY FIL X.LMD OUT=X.HEAD
All buffers from X.LMD written into Y.LMD.
$ MUTIL COPY FIL X.LMD Y.LMD 0 10
10 buffers from X.LMD written into Y.LMD.
$ MUTIL TYPE FIL X.LMD Y.LMD 10 1
Write 11th buffer of X.LMD to Y.LMD.
```

Remarks

File name I$FILCM.PPL

Created by I$FILCM.PPL

Description

CALLING STS=I$FIL_C(CV_file,CV_outfile,L_skip,L_buffers)

COMMAND COPY FILE file outfile skip buffers
Arguments/Parameters description

FILE

Routine arg. Input CHAR(*) VAR

Command par. required string replace
File name for input.

OUTFILE

Routine arg. Input CHAR(*) VAR

Command par. string replace
File name for output.

SKIP

Routine arg. Input BIN FIXED(31)

Command par. integer default=0
Optional number of buffers to be skipped.

BUFFERS

Routine arg. Input BIN FIXED(31)

Command par. integer default=100000000
Optional number of buffers to be output.

Function

Read specified input file and output GOOSY file header and data. Only standard GOOSY data formats are supported.

CREATE PROGRAM

CREATE PROGRAM file structure
/MBD /J11
/STRUCTURE
/[NO]PROGRAM
/COMPILE

PURPOSE Generates J11 stand alone programs and event data element declarations from CAMAC description file. (called in MUTIL)

PARAMETERS

file CAMAC description file. File type should be .CAM.

structure Optional name for structure of event data element. This name is used as filename for a declaration file and as structure name. If not specified, the name of the input file postfixed by '_DCL' is used. Filetype is .TXT.

/MBD/J11 Default = /MBD
Create declaration for a MBD or J11 event.

/STRUCTURE Create declaration file.

/PROGRAM Default for /MBD
Create J11 programs (only for /MBD)
The names of the programs are composed from the
name of the CAMAC file postfixed by 'Cn' where n is the crate number.
The file type is .MAC.

/COMPILE Compile and link J11 programs (only for MBD)

EXAMPLE CRE PROG TEST.CAM X$EVT /MBD/STRUC/PROG/COMP
Generates files TESTC0.MAC, TESTC1.MAC,.. and
X$EVT.TXT for the declaration of structure X$EVT.
The macro files are compiled and linked.
CRE PROG TEST.CAM X$EVT /J11/STRUC
Generates file X$EVT.TXT for the declaration of
structure X$EVT.

Description

FUNCTION The CAMAC description file as described in the harware manual is used as input to generate declarations for event data elements and J11 stand alone programs for MBD systems.

MBD A structure declaration matching SA$MBD is generated. Optionally the J11 programs to read out the CAMAC crates is generated and compiled.

J11 For single crate systems a declaration matching SA$EVENT is generated. The module names as specified in the description file are inserted in the structure.

DEBUG VME MEMORY

DEBUG VME MEMORY start end value
VMEcrate,processor ID dummy crate node
/READ/WRITE/COPY [=OPER]
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Read/write/copy VME memory.

PARAMETERS

start integer (def=0)
Start memory address (source)

end integer (def=0)
End memory address (destination)

value integer (def=0)
Value for write

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

crate Crate number

node NET node

/READ/WRITE/COPY Select read or write/read or copy.

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE DEB VME MEM

Description

FUNCTION	Read/write/copy VME memory.
File name	I$ACV_DEB_MEM.PPL
Action rout.	I$ACV_DEB_MEM
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

DETACH ANALYSIS

DETACH ANALYSIS

PURPOSE Detach data base of analysis.

REMARKS Analysis must be stopped. Dynamic lists must be detached.

Description

FUNCTION This command calls $IBUFFER to detach the analysis data base. The initialisation can be explicitely done by command INIT ANA or ATTACH ANA. Any START command calls $IBUFFER after a DETACH command. The analysis must be stopped for DETACH.

File name I$ANACM.PPL

Action rout. I$ANACM_DET

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

DETACH DYNAMIC LIST

DETACH DYNAMIC LIST dyn_list dyn_dir base node

PURPOSE detach dynamic list

PARAMETERS

dyn_list Dynamic list name specification
common required

dyn_dir Default Directory
common default:'$DYNAMIC'

base Default Data Base name
common default:'DB'

node Default node name
common default:'E'

EXAMPLE -

Caller M$DLCMD

Author H.G. Essel

File name M$ADADL.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$ADADL(CV_DYN_LIST,CV_DYN_DIR,
CV_BASE,CV_NODE)

ARGUMENTS

CV_DYN_LIST	Dynamic list name specification CHAR(*) VAR Input
CV_DYN_DIR	Default Directory CHAR(*) VAR Input
CV_BASE	Default Data Base name CHAR(*) VAR Input
CV_NODE	Default node name CHAR(*) VAR Input
FUNCTION	Detach dynamic list
REMARKS	Module is an action routine.
EXAMPLE	-

DISMOUNT TAPE

DISMOUNT TAPE device
/[NO]UNLOAD

PURPOSE	Dismount tape.
PARAMETERS	-
device	required string global replace Tape device.
/[NO] UNLOAD	switch default=/NOUNLOAD Unload tape after dismount.
EXAMPLE	DISMOUNT TAPE M1 /NOUNL

Description

FUNCTION	Dismount a tape.
File name	I$ACQ_DISMOUNT.PPL
Action rout.	I$ACQ_DISMOUNT
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	04-Jan-1988

DUMP MBD

DUMP MBD FROM=f TO=t BDO=bdo BDD=bdd
/HEXADECIMAL/DECIMAL/OCTAL/BIT
/INSTRUCTION

PURPOSE Format a MBD memory dump

PARAMETERS

FROM=f First address to be dumped, must be between 0 and 4095. NOTE, that you may enter the addresses in any radix, e.g. %O100 specifies 100 octal. The default is 0.

TO=t Last address to be dumped, must be between the value given for FROM and 4095. The default is 4095. The whole memory will be dumped if either FROM or TO is specified, but zero of repeating locations are compacted in the dump.

BDO=bdo MBD object file. If this parameter is given, the specified object file will be read and the given address range is dumped. The default file type is BDO. You may specify any file which is valid for loading with the LOAD MBD command.

BDD=bdd MBD dump file. If this parameter is given, the specified dump file will be read and the given address range is dumped. The default file type is BDD. The dump files are in general created with the STORE MBD command. NOTE: If neither BDO nor BDD are specified, the command will show the memory contents of the currently active MBD.

/HEXADECIMAL Dump the memory contents with hexadecimal radix.

/DECIMAL Dump the memory contents with decimal radix.

/OCTAL Dump the memory contents with octal radix. This is the default.

/BIT Dump the memory contents with binary radix.

/INSTRUCTION Dump the memory contents as disassembled instructions. Each line contains the address of the instruction and the instruction itself in octal

radix, the CNAF interpretation and the mnemonics of the disassembled instruction. Note, that the CNAF numbers are in decimal radix, whereas all other numbers are in octal radix !!

FUNCTION This command dumps the contents of the MBD memory, an MBD object file or a MBD memory dump file in either hexadecimal, decimal, octal or binary radix. Note, that the addresses in the first column are always printed in octal radix. Repetive lines are suppressed in order to avoid long output indicating a zeroed memory.

Action rout. I$MCDMM

Author Walter F.J. Mueller

Remarks

File name I$MCDMM.PPL

Dataset -

REMARKS -

EXAMPLE DUMP MBD

Description

CALLING @CALL I$MCDMM(I_FROM,I_TO,C_BDO,C_BDD,C_MODE);

ARGUMENTS

I_FROM BIN FIXED(15) [INPUT]
First address to be dumped, between 0 and 4095.

I_TO BIN FIXED(15) [INPUT]
Last address to be dumped, between I_FROM and 4095.

C_BDO CHAR(*) VAR [INPUT]
Name of a MBD object file, the default file type is .BDO.

C_BDD CHAR(*) VAR [INPUT]
Name of a MBD dump file, the default file type is .BDD.

C_MODE CHAR(*) VAR [INPUT]
Set of mode qualifiers:

/DECIMAL	Write in decimal radix.
/HEXADECIMAL	Write in hexadecimal radix.
/OCTAL	Write in octal radix.
/BIT	Write in binary radix.
/INSTRUCTION	Write intruction mnemonics.

FUNCTION This procedure dumps either the memory of the MBD directly or the contents of an object or dump file. If both C_BDO and C_BDD are empty strings, the current MBD memory will be shown. If one of the file names is given, it will be read and dumped.

REMARKS -

EXAMPLE @CALL I$MCDMM(0,4095,",",'/DECIMAL');

DUMP STARBURST

DUMP STARBURST FROM=f TO=t C=c N=n TSK=file
/HEXADECIMAL/DECIMAL/OCTAL/BIT

PURPOSE Format a STARBURST memory dump through MBD

PARAMETERS

FROM=f First address to be dumped, must be between 0 and and the memory size. The address must be specified as a byte address (as all addresses in a PDP-11), but it is always rounded to a word boundary. NOTE, that you may enter the addresses in any radix, e.g. %O100 specifies 100 octal. The default is 0.

TO=t Last address to be dumped, must be between the value given for FROM and the memory size.

C=c Crate number of the STARBURST to be accessed. The default is Crate 1.

N=n Station number of the STARBURST to be accessed. The default is Station 23.

TSK=file Task image file. If this parameter is given, the specified image file will be read and the given address range is dumped. The default file type is TSK. You may specify any file which is valid for loading with the LOAD STARBURST command. NOTE: If TSK is not specified, the command will show the memory contents of the STARBURST in crate C and station N.

/HEXADECIMAL Dump the memory contents with hexadecimal radix.

/DECIMAL Dump the memory contents with decimal radix.

/OCTAL Dump the memory contents with octal radix. This is the default.

/BIT Dump the memory contents with binary radix.

FUNCTION This command dumps the contents of a STARBURST memory or a image file either hexadecimal, decimal, octal or binary radix. Note,

that the addresses in the first column are always printed in octal radix. Repetive lines are suppressed in order to avoid long output indicating a zeroed memory.

EXAMPLE DUMP STARBURST

Action rout. I$MCDMS

Author Walter F.J. Mueller

Remarks

File name I$MCDMS.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$MCDMS(L_FROM,L_TO,I_C,I_N,C_TSK,C_MODE);

ARGUMENTS

L_FROM BIN FIXED(31) [INPUT]
Lower dump address limit, between 0 and the memory size.

L_TO BIN FIXED(31) [INPUT]
Upper dump limit, between L_FROM and the memory size.

I_C BIN FIXED(31) [INPUT]
Crate number of STARBURST to be accessed.

I_N BIN FIXED(31) [INPUT]
Station number of STARBURST to be accessed.

C_TSK CHAR(*) VAR [INPUT]
Name of Task image file to be dumped. If omitted, the CAMAC module specified with I_C and I_N will be accessed.

C_MODE CHAR(*) VAR [INPUT]
Mode qualifier set:

/DECIMAL Write in decimal radix.

/HEXADECIMAL	Write in hexadecimal radix.
/OCTAL	Write in octal radix.
/BIT	Write in binary radix.

FUNCTION This procedure dumps either the memory of a STARBURST directly or the contents of an image file. If both C_TSK is an empty strings, the current STARBURST memory will be shown. Otherwise the image file is read and dumped.

REMARKS -

EXAMPLE @CALL I$MCDMS(0,4095,1,23,",'/DECIMAL');

EXECUTE VME

EXECUTE VME command VMEcrate,processor ID
dummy subcrate node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Execute command on remote VME processor

PARAMETERS

command command string (counted ASCII, zero terminated)

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

subcrate subcrate

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE EXE VME "START RUN" /AEB

Description

FUNCTION Execute command on remote VME processor

File name I$ACV_EXE_VME.PPL

Action rout. I$ACV_EXE_VME

Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

FOREIGN ACQUISITION

FOREIGN ACQUISITION string longword
/X /Y /Z [=QUAL]
/[NO]SWITCH

PURPOSE Whatever

PARAMETERS

string string replace default="
description

longword integer default=8192 range=(1,24576)
description

QUAL qualifier default=/X

/X	decription
/Y	decription
/Z	decription

/SWITCH negatable switch default=/SWITCH
description
+Command

Module Command : INITIALIZE ACQUISITION mailbox size count
in_buffers out_buffers
node command data
/MBD /J11 /FILE/FOREIGN/VME
/PAGE /BYTE /KBYTE
/MBX

PURPOSE Init data taking

PARAMETERS

mailbox string replace
String to build mailbox name:
GOOSY_mailbox_1,2,3

Three mailboxes are created.
Default is the environment name. The name must be unique. This is ensured by using the environment name.

size integer replace default=8192
Size of listmode data buffers in bytes, must be between 512 and 28672. The size should be a multiple of 512 or even better a multiple of 4096. Together with the /PAGE /KBYTE qualifier this quantity can be specified in different units than bytes. If zero, defaults to 16384 for VME, 8192 others.

count integer default=8
Number of listmode data buffers, must be between 4 and 32. The default is normally adequate.

in_buffers integer default=4
Number of buffers queued to MBD.
The default is normally adequate.

out_buffers integer default=0
Number of buffers hold before writing to the output device is started. The default is normally adequate.

node string replace
Node of J11 (only needed for /J11)

command string replace default=CMDERR
Command component of J11 (/J11 only)
The default is adequate.

data string replace default=TMR11S
Data component of J11 (/J11 only)
The default is adequate.

/MBX Open input mailbox. String to build mailbox name:
GOOSY_mailbox_I1

/MBD CAMAC interface is MBD

/J11 CAMAC interface is J11

/FILE Input from file (use OPEN-CLOSE FILE)

/FOREIGN Input from unknown source.

/PAGE Buffer size in pages (512 bytes)

/BYTE	Buffer size in bytes
/KBYTE	Buffer size in Kbytes (1024 bytes)
EXAMPLE	INIT ACQU ANNA SIZE=8192 COUNT=12 IN_BUF=6 creates mailboxes GOOSY_ANNA_1,2,3

Description

FUNCTION	This procedure initializes the buffer pool used for listmode data. The given number of buffers is allocated and locked in the workingset. NOTE: The number and size of buffers cannot be changed afterwards whereas the queuing parameters may be modified. In J11 mode the links to the J11 are opened. he mailboxes are created. The size of the mailboxes ust be 8192 for the J11 input. This value is efaulted with the /J11 qualifier. With /MBD the ailbox size default is 4096.
File name	I$ACQ_INI_ACQ.PPL
Action rout.	I$ACQ_INI_ACQ
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

INITIALIZE ANALYSIS

INITIALIZE ANALYSIS base1 base2 base3
/[NO]ANALYSIS
/[NO]UNPACK
/[NO]PACK
/[NO]START
/[NO]STOP
/[NO]BASE

PURPOSE Reinitialize analysis (Analaysis must be stopped)

base1 string
Data base to attached.

base2 string
Data base to attached.

base3 string
Data base to attached.

/NOANAL Disable calling of loaded analysis routine.

/ANAL Initialize and enable loaded analysis routine. The module must have been loaded with
LOAD MOD ANAL anal ianal /ANAL

/NOUNPACK Disable calling of loaded unpack routine.

/UNPACK Initialize and enable loaded unpack routine. The module must have been loaded with
LOAD MOD ANAL unpack iunpack /UNPACK

/NOPACK Disable calling of loaded pack routine.

/PACK Initialize and enable loaded pack routine. The module must have been loaded with
LOAD MOD ANAL pack ipack /PACK

/NOSTART Disable calling of loaded START routine.

/START	Initialize and enable loaded start routine. The module must have been loaded with LOAD MOD ANAL start istart /START
/NOSTOP	Disable calling of loaded STOP routine.
/STOP	Initialize and enable loaded stop routine. The module must have been loaded with LOAD MOD ANAL stop istop /STOP
/[NO] BASE	[No]base available

Description

FUNCTION	This command calls I$ANACM_DET to detach the analysis. Then it checks weather the specified modules have been loaded. START and STOP routines are initialized, if entries have been specified. Then I$ANACM_ATT is called to reinitialyze the analysis. The /NO... switches switch off the calling of the loaded module. Then the default modules are called.
File name	I$ANACM.PPL
Action rout.	I$ANACM_INIT
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

INITIALIZE CAMAC

INITIALIZE CAMAC VMEcrate,processor ID dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Initialize CAMAC

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE

Description

FUNCTION Send INIT CAMAC command .

File name I$ACV_INI_CAM.PPL

Action rout. I$ACV_INI_CAM

Dataset -

Version 1.01

Author H.G.Essel

Last Update 16-feb-1989

LOAD J11

LOAD J11 file events
/KEEP /COMPRESS

PURPOSE	Load CAMAC module table into J11
PARAMETERS	
file	string replace default=J11.CAM File containing module specifications. See Hardware Manual for description.
events	integer replace default=0 Maximum number of events to be filled in a buffer. A value of 0 (=default) means as many as possible. Acquisition must be initialized. STOP the ACQUISITION before loading the J11.
/COMPRESS	Suppress zeros.
/KEEP	Keep buffers in J11
EXAMPLE	LOAD J11 mymod.CAM

Description

FUNCTION	Loads module description table to J11.
File name	I$ACQ_LOA_J11.PPL
Action rout.	I$ACQ_LOA_J11
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	24-feb-1987

LOAD LRS_2365

```
LOAD LRS_2365 file C=c N=n
    /[NO]LOG
    /[NO]DUMP
```

PURPOSE Load definitions in a LRS 2365 logic matrix

PARAMETERS

file Name of a definition file, the default file type is DAT.

C=c Crate number of the module to be loaded.

N=n Station number of the module to be loaded.

/[NO] LOG Logs the contents of the definition file on SYS$OUTPUT. The definition file is just copied to SYS$OUTPUT while they are read and parsed.
/NOLOG: Don't log definition file, this is the default.

/[NO] DUMP Dumps the contents of the LRS 2365 module found after the down load. The programming words PW 0 to 17 are shown in binary radix. For their interpretation look in the LRS manual.
/NODUMP: Don't dump programming words, this is the default.

FUNCTION This procedure reads the definition file and generates the programming words needed to load the LRS 2365 logic matrix. This unit is an eight fold programmable overlap coinzidence with 16 inputs. Each input can be enable, disabled or inverted for each of the 8 channels, the outputs may be inverted. This allows to perform AND as well as OR operations. The definition file must have the format: The character of the line must be '!','A','O','T'.
A line beginning with a '!' is a comment and will be ignored.
A line beginning with an 'A' or 'O' indicates an AND or OR operation. There may be up to 8 lines beginning with 'A' or 'O', one for each output channel. The rest of the line may contain up to 16 '+',

'-' or 'X' indicating whether an input is to be used directly or inverted or to be ignored.
A line beginning with a 'T' defines the TEST partern. The rest of the line may contain up to 16 '0' or '1' seperated by blanks. A sample input file may look like:

```
! Some comments
!
! 1 1 1 1 1 1 1
! 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
!
A + + X X X X X X X X X X X X X X
A X + + X X X X X X X X X X X X X
O + + + X X X X X X X X X X X X X
!
A X X X + - X X X X X X X X X X X
A X X X X + X X X X X X X X X X X
A X X X X X + X X X X X X X X X X
A X X X X X X + X X X X X X X X X
A X X X X X X X + X X X X X X X X
```

This results in:

Out(0) = In(0) .AND. In(1)
Out(1) = In(1) .AND. In(2)
Out(2) = In(0) .OR. In(1) .OR. In(2)
Out(3) = In(3) .AND. .NOT. In(4)
Out(4) = In(4)
Out(5) = In(5)
Out(6) = In(6)
Out(7) = In(7)

EXAMPLE LOAD LRS2365 MYCOINZ 1 1

Action rout. I$LDLRS_2365

Author Walter F.J. Mueller

Remarks

File name I$LDLRS_2365.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$LDLRS_2365(C_FILE,I_C,I_N,C_LOG,C_DUMP);

ARGUMENTS

C_FILE CHAR(*) VAR [INPUT]
Name of the definition file.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be loaded.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be loaded.

C_LOG CHAR(*) [INPUT]
LOG qualifier set:

/LOG	Log definition file.
/NOLOG	Don't log definition file.

C_DUMP CHAR(*) [INPUT]
DUMP qualifier set:

/DUMP	Dump programming words.
/NODUMP	Don't dump programming words.

FUNCTION Reads the definition file and load a LRS 2365 module. For details look in the command description.

REMARKS -

EXAMPLE @CALL I$LDLRS_2365('MYDEF',1,1,",");

LOAD MBD

LOAD MBD file
/EXECUTIVE

PURPOSE Load microcode in MBD, either executive or a channel program.

PARAMETERS

file Name of the compiled MBD program. The default file type is BDO (for Branch driver object code). The program must have been written with the MBD macro sets MBDGLO and MBD2PG and must have been compiled with the VMS/RSX cross assembler. It must furtheron contain the propper header indicating start address, length and channel. For details look in the function description.

/EXECUTIVE Enables loading of the executive. The executive is the code for the MBD channel 7. It performs common functions like load of other channels and read back of the MBD memory. The executive code must be loaded before any other code after a boot, but a reload of it later on will erase all other channel codes.

FUNCTION This procedure loads the MBD memory with program code which has been compiled with the VMS/RSX compatibility mode assembler in absolute mode. The channel code must have been compiled with the MBDGLO and MBD2PG macro sets in absolute mode. The program must be prefixed by a header which contains information about:

the program length (in 16 bit words)
the load and init start address
the channel, for which the program should execute

A sample program may look as follows:

```
.TITLE CNAF-PROGRAM
.LIST TTM
.ENABLE ABSOLUTE
.MCALL MBDGLO,MBD2PG
MBDGLO
MBD2PG
```

```
        .MCALL TPAGE,ENTRY,CONST,VARIA,COFCNA,FILL
        .MCALL
CAMWRT,CAMWRC,CAMFNC,CAMRD,CAMW,CAMVEC
    ;
    $=400 ; fix the start and init address
    JCSTA=$  ; begin marker
                    ; set up 4 word header block with:
                    ; length (including header!)
                    ; a 0 (was function modifier)
                    ; start and init address
                    ; channel number (0..7)
        .WORD JCEND-JCSTA+4 ; length
        .WORD 0
        .WORD JCSTA ; start & init
        .WORD 0 ; channel number 0
        .....
        any MBD code, for an example look in
            GOO$IO:ESONE.MBD
        .....
        JCEND=$  ; end marker
        .END
```

REMARKS If a MBD code will be loaded into a MBD where another MBD code was loaded before, the new code be shorter (or equal) in memory than the old one. Otherwise use the RELEASE MBD CHANNEL command before loading the new code.

EXAMPLE LOAD MBD MYPROG

Action rout. I$MCLDM

Author Walter F.J. Mueller

Remarks

File name I$MCLDM.PPL

Dataset -

Description

CALLING @CALL I$MCLDM(C_FILE,C_MODE);

ARGUMENTS

C_FILE	CHAR(*) VAR [INPUT] File name of the compiled MBD program.
C_MODE	CHAR(*) VAR [INPUT] Executive qualifier, enables to load executive.
FUNCTION	Load the MBD executive or channel code.
REMARKS	-
EXAMPLE	@CALL I$MCLDM('MYPROG','');

LOAD MODULE ACQUISITION

LOAD MODULE ACQUISITION image module init
/START/STOP [=TYPE]

PURPOSE Load module from sharable image.

PARAMETERS

image string
name of sharable image for routines.

module string
name of routine.

init string
name of initialization entry of routine.

/START switch
The module is called at the beginning of
START ACQUIS

/STOP switch
The module is called at the end of
STOP ACQUIS
The modules must be linked into a sharable image by DCL command LSHARIM. The initialization entry is called.

EXAMPLE LOAD MOD ACQU MYSHARE X$START $XSTART /START

Description

FUNCTION Loads a sharable image and the specified module. The sharable image must be linked by command:
LSHARIM (see help).

File name I$ACQ_LOA_MOD.PPL

Action rout. I$ACQ_LOA_MOD

Dataset -

Version	1.01
Author	H.D.Essel
Last Update	12-APR-1985

LOAD MODULE ANALYSIS

LOAD MODULE ANALYSIS image module init
/START/STOP/UNPACK/PACK/ANAL [=TYPE]

PURPOSE Load module from sharable image.

PARAMETERS

image required character
name of sharable image for routines.

module required character
name of routine.

init character
name of initialization entry of the routine.
Arguments for /UNPACK and /PACK:
POINTER,BIN FIXED(31)
Pointer to event data element and length in bytes.
These arguments are zero until set by the commands
SET EVENT INPUT (for unpack)
SET EVENT OUTPUT (for pack)
Arguments for /START /STOP and /ANAL:
no arguments.

/START switch
The module is called at the beginning of
START INPUT ...
without arguments.

/STOP switch
The module is called at the end of
STOP INPUT ...
without arguments.

/UNPACK switch
The module is called for event unpacking
with one argument:
The pointer to GOOSY buffer or event, depending
on the mode SET ANAL /BUFFER or /EVENT

/PACK	switch The module is called for event packing with one argument: The pointer to GOOSY output buffer.
/ANAL	switch The module is called after event unpacking without arguments.
EXAMPLE	$ LSHAR X$ANAL,X$START,X$STOP MYSHARE /GROUP $ GOOSY STOP INPUT MAIL LOAD MOD ANAL MYSHARE X$ANAL $XANAL /ANAL LOAD MOD ANAL MYSHARE X$START /START LOAD MOD ANAL MYSHARE X$STOP /STOP INIT ANAL /ANAL/START/STOP
NOTE	The modules must be linked into a sharable image by DCL command LSHAR. The initialization entries are called by the INIT ANAL command.

Description

FUNCTION	Loads a sharable image and the specified module. The sharable image must be linked by command: LSHARIM (see help). The analysis must be stopped. The modules must be enabled by INIT ANALYSIS to be called. Previous loaded modules are not called any longer.
File name	I$ANACM.PPL
Action rout.	I$ANACM_LOA_MOD
Dataset	-
Version	1.01
Author	H.D.Essel
Last Update	12-APR-1985

LOAD STARBURST

LOAD STARBURST file C=c N=n
/[NO]HALT
/BOOT/INIT

PURPOSE Load a system or task image in the STARBURST memory.

PARAMETERS

file Name of a RSX11-M task or system image. The default file type is TSK.

C=c Crate number of the STARBURST to be loaded.

N=n Station number of the STARBURST to be loaded.

/[NO] HALT The CPU is explicitely halted before any load operation is done. This is the default.
/NOHALT: The CPU is not halted before the load operation. this may be usefull for the load of a data partition or of a task partition.

/BOOT The CPU is booted after the load at the start address found in the task image label. This is done by writing a JMP @#<startaddress> instruction in the memory locations 0 and 2, setting the powerup interrupt address to 0 (in location 24,26) and by simulating a power failure.

/INIT The CPU is halted after the load and put into console ODT mode. Type <address>G on the local console to start program execution. This can be used to start at an address different from the system start address.

FUNCTION This procedure loads a system or task image into the memory of a STARBURST processor. The processor is halted (if /NOHALT is not specified) before the memory is accessed in order to avoid race conditions with programs still running in the STARBURST. Than the image file is loaded and read back to verify a propper load. Note, that only first 128 kbyte are accessible for this command.
The CPU is either put in console ODT mode (/INIT) or booted at the start address found in the image label (/BOOT). A STARBURST

bootable system image should be task builded with the following options:

```
$ MCR TKB
<image>.TSK/-HD,<image>/-SP=<object modules>
/
STACK=0
//
```

The DCL command LINKJ11 may be used to link a J11 task program.

EXAMPLE LOAD STARBURST MYPROG C=1 N=23 /BOOT

Action rout. I$MCLDS

Author Walter F.J. Mueller

Remarks

File name I$MCLDS.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$MCLDS(C_FILE,I_C,I_N,C_HALT,C_START);

ARGUMENTS

C_FILE CHAR(*) VAR [INPUT]
File name of the compiled MBD program.

I_C BIN FIXED(15) [INPUT]
Crate number of the STARBURST to be loaded.

I_N BIN FIXED(15) [INPUT]
Station number of the STARBURST to be loaded.

C_HALT CHAR(*) VAR [INPUT]
Halt qualifier set:

/HALT	Halt CPU before load.
/NOHALT	Load without halting CPU.

C_START CHAR(*) VAR [INPUT]
Boot qualifier qualifier set:

/INIT	Init CPU, activate console ODT.
/BOOT	Boot CPU via power fail simulation.

FUNCTION This procedure loads a system image into a STARBURST processor and boots it.

REMARKS -

EXAMPLE @CALL I$MCLDS('MYPROG',1,23,",");

LOAD VME PROGRAM

LOAD VME PROGRAM file procrate processor id subcrate
node
/TABLE
/[NO]LOAD
/RESET
/FEP/EB/ROP
/CVI/CVC/CAV/AEB/VME/EBI
/USER
/[NO]SYNC
/[NO]SYSTEM

PURPOSE Load programs into VME processors

PARAMETERS

file string
File containing exec code produced by PCP.
or @file containing description file. File type VMEP is defaulted. For this case only qualifier /USER/SYSTEM is used.

procrate integer
VME processor crate NOT USED

processor integer
Processor offset 0-13 NOT USED

id integer NOT USED
Processor ID

node J11 node (default=J11B) NOT USED

/TABLE Load readout tables from same file.

/LOAD Load modules (=default)

/RESET Call I$ACV_RES_VME first to reset processor.
The following qualifiers are not yet active:

/FEP Processor is FEP

/EB	Processor is EB
/ROP	Processor is ROP
/CVI	Processor controls CAMAC crate with CVI processor
/CVC	Processor controls CAMAC crate with CVC processor
/CAV	Processor controls CAMAC crate with controller
/AEB	Processor controls Fastbus crate with AEB
/EBI	Processor controls EBI memory with foreign equipm.
/VME	Processor controls VME crate.
/SYNC	FEP runs in asynchron mode.
/USER	Load user routine specified in file.
/SYSTEM	Load system executive.
EXAMPLE	LOAD VME PROG @setup

Description

FUNCTION	Loads programs to VME processors. When the file is the exec program, either processor ID and subcrate or VME crate, offset and subcrate must be specified. These values must match with previously loaded processor.
File name	I$ACV_LOA_VME_P.PPL
Action rout.	I$ACV_LOA_VME_P
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

Syntax

The description file contain lines with the following syntax:

lines <object> <specification>,<specification>,...

object PROCESSOR — MODULE

specification <key>=<value>

value string — number — (<spec>,<spec>,...) —
(number,number,...)
Lines can be continued by a hyphen at the end.
Comments are preceded by exclamation mark. Other files are included using a line @ <newfile>

PROCESSOR Processor specification lines specify
1. The software to be loaded
2. The Bus and processor types
BRANCH=(CRATE=c,OFFSET=p,ID=id),
CONTROL=CAV—CVI—CVC—AEB—VME—EBI,
TYPE=FEP—EB—ROP,
MODE=SYNC—ASYNC, default=SYNC
CRATE=c, not used
SYSTEM=file, should be logical name
USER=file not used
All asynchron FEP's must be behind the synchron FEP's. First FEP is master and sets/clears GO bit in trigger module.

MODULE Module specification lines specify
1. the location of a hardware module
2. the readout, reset or init information.
These lines are ignored.

LOAD VME TABLE

LOAD VME TABLE file trigger VMEcrate,processor ID
subcrate node log
/[NO]LOAD
/LOG
/OVER
/FEP/EB
/ROP/ROC/AEB/VME

PURPOSE Load tables into VME processors. See also I$VMETAB

PARAMETERS

file string
File containing description file.

trigger integer
Trigger number for the table.

VMEcrate,processor integer array
VMEcrate,Processor offset 0-13

ID integer
Processor ID

node ethernet node (default=ETH_02)

logfile optional log file

/LOG Output tables contents

/LOAD Load tables (=default)
The following qualifiers are not yet active:

/OVER Overwrite processor crate, offset in file by values specified in command.

/FEP Processor is FEP

/EB Processor is EB

/ROP Processor controls CAMAC crate with processor

/ROC	Processor controls CAMAC crate with controller
/AEB	Processor controls Fastbus crate with AEB
/VME	Processor controls VME crate.
EXAMPLE	LOAD VME TAB @MYSETUP.VMEP

Description

FUNCTION	Loads tables to VME processors.
File name	I$ACV_LOA_VME_T.PPL
Action rout.	I$ACV_LOA_VME_T
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

Syntax

The description file contain lines with the following syntax:

lines	<object> <specification>,<specification>,...
object	PROCESSOR — MODULE
specification	<key>=<value>
value	string — number — (<spec>,<spec>,...) — (number,number,...) Lines can be continued by a hyphen at the end. Comments are preceded by exclamation mark. Other files are included using a line @ <newfile>
PROCESSOR	Processor specification lines specify 1. The software to be loaded 2. The Bus and processor types 3. The branch ID (for crate,offset) BRANCH=(CRATE=c,OFFSET=p,ID=id), CONTROL=ROP—ROC—AEB—VME, Only the branch specification is used.

MODULE Module specification lines specify
1. the location of a hardware module
2. the readout, reset or init information.

BRANCH=(CRATE=c,OFFSET=p)—(ID=id),
CONTROL=CAV—EBI—CVI—AEB—VME,
NAME=name, NOT YET USED
TYPE=type,CRATE=c,STATION—N=n,SUBADDRESS—A=a,
INIT=(FUNCTION=f,REPEAT=r,EXEC=e,DATA=d,A=a),
READ=(FUNCTION=f,REPEAT=r,EXEC=e,DATA=d),
RESET=(FUNCTION=f,REPEAT=r,EXEC=e),
CHANNEL=c,DATA=(v,v,...),DATA=@filespec
For 16 <= functions <= 23 a data value must be
specified with INIT/READ

FASTBUS_pedestals

Fastbus pedestals may be written in a text file with four numbers per line separated by commas.

low thresh,low ped,high thresh,high ped

This file must be specified with

DATA=@file

in the fastbus module line.

EXECUTION_codes

valid values are

ON*LY	execute only
X	execute and check X-response
Q	execute and check Q-response
Q1	execute until Q=1
Q0	execute until Q=0
XQ	execute and check X- and Q-response
XQ1	execute and check X until Q=1
XQ0	execute and check X until Q=0

CAMAC_types

the TYPE keyword is optional, can be replaced and
overwritten by INIT/READ/WRITE)

ST*ANDARD for LRS2248a, Ortec AD811, AD1000 without FIFO, Borer scalers, ...

SI*4418 Silena T/V/Q ADC's

AD8*000 same as SI4418

CS*D24 Wenzel counter

AD1*000 FIFO on

PU*1000 same as AD1000 with FIFO on

LRS4302

LRS4300 Fera 16 channel ADC

LRS2*261 image chamber analyzer

FASTBUS_types

supported are

LRS1872 LeCroy 64-channel TDC (12 bit)

LRS1875 LeCroy 64-channel TDC (12/15 bit)

LRS1882 LeCroy 96-channel ADC (12 bit)

LRS1885 LeCroy 96-channel ADC (12/15 bit)

KSCF432 Kinetic Systems 64-channel TDC

STR136 Struck 64-bit pattern unit

STR200 Struck 32-channel scaler (32 bit)

MOUNT TAPE

MOUNT TAPE device label blocksize density
/INITIALIZE
/DISMOUNT
/TK50 /TK70 /EXABYTE

PURPOSE Mount RMS tape.

PARAMETERS -

device required string global replace default="
Tape device.

label string
Label of tape used for initialization.

blocksize integer replaced default=24576
Blocksize should be multiple of buffersize (8192)

density integer replaced default=6250
Tape density 800, 1600 or 6250 (depends on tape drive).

/INITIALIZE switch
Initialize tape before mounting. Write specified label to tape.

/DISMOUNT switch
Dismount a currently mounted tape and unload it. After that, the new tape must be mounted on the device. A prompt will give you the chance to do that and to continue.

/TK50 switch
Tape is TK50.

/TK70 switch
Tape is TK70.

/EXABYTE switch
Tape is EXABYTE.

EXAMPLE MOUNT TAPE M1 xx001 /INIT

Description

FUNCTION	Mount a tape and initialize it optionally. GOOSY writes ANSI labeled tapes. The tape label is written to the tape during the initialization. If the tape is already initialized, the tape label is read and stored internally. It is used as default for GOOSY file headers in the START OUTPUT FILE command. With the /DISMOUNT qualifier a currently mounted tape is dismounted. Then a prompt is output to wait untill the new tape is mounted on the device.
File name	I$ACQ_MOUNT.PPL
Action rout.	I$ACQ_MOUNT
Version	1.01
Author	H.G.Essel
Last Update	04-Jan-1988

OPEN FILE

OPEN FILE filename device directory headerfile
/[NO]HEADER

PURPOSE	Open file for data input stream.
PARAMETERS	-
filename	required string replace Name of listmode data file used for input.
device	string replaced Default device.
directory	string replaced Default directory.
headerfile	string Optional file to write file header.
/HEADER	switch default=/HEADER File has GOOSY header.
EXAMPLE	OPEN FILE j11.lmd DEV=M0:
NOTE	The acquisition must be initialized with /FILE

Description

FUNCTION	Opens a file for data input. The input is started by START ACQUIS. The data is processed as from other input channels. This input is used for multiple parallel working analysis programs on different nodes.
NOTE	The acquisition must be initialized with /FILE
File name	I$ACQ_OPE_FIL.PPL
Action rout.	I$ACQ_OPE_FIL
Dataset	-

Version	1.01
Author	H.G.Essel
Last Update	20-AUG-1987

OPEN OUTPUT FILE

OPEN OUTPUT FILE file size number
device directory
headerinput headeroutput
/PROMPT
/EDIT
/AUTOMATIC
/ALLOCATE
/PAGE /BYTE /KBYTE /BUFFER

PURPOSE Open list mode dump file

PARAMETERS

file required string replace
Name of the file to be opened. The default file type is .LMD. Device and directory are defaulted from the parameters DEVICE and DIRECTORY. Keep in mind filename comventions for IBM.

size integer replace default=35000
Size limit of the file to be written in Kbytes. Other units can be selected by /PAGE/BYTE/BUFFER

number integer replace default=1000
Number of automatically written files (/AUTO)

device string replace
Default device

directory string replace
Default directory

headerinput string replace default="
Optional file to read file header.

headeroutput string replace default="
Optional file to store file header.

/PROMPT Prompt file header information

/EDIT	Edit file header (Headerinput required).
/AUTOMATIC	A three digit number is appended to the file name. If the file is filled, a new file is opened. The number is incremented.
/ALLOCATE	Preallocate file (disk only)
/PAGE	File size in pages (512 bytes)
/BYTE	File size in bytes
/KBYTE	File size in Kbytes (1024 bytes =default)
/BUFFER	File size in buffers
EXAMPLE	OPEN OUT FIL RUN026 DEV=M0:
NOTE	This command is called by START OUTPUT FILE which is the command one should use.

Description

FUNCTION	Open list mode file. If one wants to send the output files to the IBM, the filenames must follow some conventions: Maximal length 25 char (including type) Maximal 8 char or 7 digits between two _ File type is .LMD
File name	I$ACQ_OPE_LMD.PPL
Action rout.	I$ACQ_OPE_LMD
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

PATCH MBD

PATCH MBD ADDRESS=a VALUE=v C=c N=n A=a F=f
/NOCONFIRM

PURPOSE	Patch a MBD memory location
PARAMETERS	
ADDRESS=a	MBD memory address to be modified, be between 0 and 4095. NOTE, that you may enter the addresses in any radix, e.g. %O100 specifies 100 octal. This is a required parameter
VALUE=v	New value for memory location ADDRESS.
C=c	Crate number, if new value is in CNAF format.
N=n	Station number, if new value is in CNAF format.
A=a	Subaddress, if new value is in CNAF format.
F=f	Function code, if new value is in CNAF format. NOTE: Either VALUE or C must be specified.
/NOCONFIRM	Will bypass the confirmation question if specified. The default is the show the address, new and old value in decimal and octal radix and to request a confirmation.
FUNCTION	This procedure replaces the value in the MBD memory location ADDRESS by VALUE or the CNAF code specified with C,N,A and F.
EXAMPLE	PATCH MBD 234 12345
Action rout.	I$MCPAM
Author	Walter F.J. Mueller

Remarks

File name	I$MCPAM.PPL
Dataset	-
REMARKS	-

Description

CALLING @CALL I$MCPAM(I_ADDR,I_VALUE,I_C,I_N,I_A,I_F,
C_NOCONF,B_DP);

ARGUMENTS

I_ADDR BIN FIXED(15) [INPUT]
Address to be modified, must be between 0 and 4095.

I_VALUE BIN FIXED(15) [INPUT]
New value, will be used if not defaulted.

I_C BIN FIXED(15) [INPUT]
Crate number for a value specification as CNAF.
Will be used if not defauled. NOTE, that either I_VALUE or I_C must be specified explicitely.

I_N BIN FIXED(15) [INPUT]
Station number for a value specification as CNAF.

I_A BIN FIXED(15) [INPUT]
Subaddress for a value specification as CNAF.

I_F BIN FIXED(15) [INPUT]
Function code for a value specification as CNAF.

C_NOCONF CHAR(*) VAR [INPUT]
/NOCONFIRM qualifier. The confirmation question is skipped if this parameter is nonblank.

B_DP BIT(*) ALIGNED [INPUT]
Default pattern as passed by the $DP pseudo argument

FUNCTION This procedure replaces the value in the MBD memory location I_ADDR by I_VALUE or the CNAF code specified with I_C,I_N,I_A and I_F.

REMARKS -

EXAMPLE @CALL I$MCPAM(243,1234,0,0,0,0,",'00111100'B);

PATCH STARBURST

PATCH STARBURST ADDRESS=a VALUE=v C=c N=n
/NOCONFIRM

PURPOSE Patch a STARBURST memory location

PARAMETERS

ADDRESS=a Address to be modified. The address is to be specified as a BYTE address but has to be word aligned (an even number) and in the range 0 to 131070 (the lower 64k words). NOTE, that you may enter the addresses in any radix, e.g. %O100 specifies 100 octal. This is a required parameter

VALUE=v New value for memory location ADDRESS.

C=c Crate number of the STARBURST to be loaded.

N=n Station number of the STARBURST to be loaded.

/NOCONFIRM Will bypass the confirmation question if specified. The default is the show the address, new and old value in decimal and octal radix and to request a confirmation.

FUNCTION This procedure replaces the value in the STARBURST memory location ADDRESS by VALUE.

EXAMPLE PATCH STARBURST %O620 1234 1 23

Action rout. I$MCPAS

Author Walter F.J. Mueller

Remarks

File name I$MCPAS.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$MCPAS(L_ADDR,I_VALUE,I_C,I_N,
C_NOCONF);

ARGUMENTS

L_ADDR BIN FIXED(31) [INPUT]
Address to be modified. The address is to be specified as a BYTE address but has to be word aligned (an even number) and in the range 0 to 131070 (the lower 64k words).

I_VALUE BIN FIXED(15) [INPUT]
New value, will be used if not defaulted.

I_C BIN FIXED(15) [INPUT]
Crate number of the STARBURST to be patched.

I_N BIN FIXED(15) [INPUT]
Station number of the STARBURST to be patched.

C_NOCONF CHAR(*) VAR [INPUT]
/NOCONFIRM qualifier. The confirmation question is skipped if this parameter is nonblank.

FUNCTION This procedure replaces the value in the STARBURST memory location L_ADDR by I_VALUE. The STARBURST is located in crate I_C and station I_N.

REMARKS -

EXAMPLE @CALL I$MCPAS(610,1234,1,23,");

RELEASE MBD CHANNEL

RELEASE MBD CHANNEL channel_no

PURPOSE	Release MBD channel to allow loading of new code
PARAMETERS	
CHANNEL_NO	Number of the MBD channel to be released
FUNCTION	Release a MBD channel. If a channel was used already and a certain code was loaded into the MBD for this channel, a new code for this channel can only be loaded if its size is smaller than or equal to the size of the old code. To allow loading of larger code on has to 'release' the channel first.
REMARKS	The command should only be used if a MBD channel code could not be loaded.
EXAMPLE	REL MBD CHAN 4
Action rout.	I$MCRMC
Author	M. Richter

Remarks

File name	I$MCRMC.PPL
Dataset	-

Description

CALLING	@CALL I$MCRMC(I_channel_no);
ARGUMENTS	
I_channel_no	I Number of MBD channel to be released. The channel number might be between 0 and 6 BIN FIXED(15)

FUNCTION Release a MBD channel. If a channel was used already and a certain code was loaded into the MBD for this channel, a new code for this channel can only be loaded if its size is smaller than or equal to the size of the old code. To allow loading of larger code on has to 'release' the channel first.

REMARKS -

EXAMPLE @CALL I$MCRMC(4);

RESET ACQUISITION

RESET ACQUISITION

EXAMPLE RESET ACQU

Description

FUNCTION

File name	I$ACQ_RES_ACQ.PPL
Action rout.	I$ACQ_RES_ACQ
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

RESET CAMAC

RESET CAMAC VMEcrate,processor ID dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Reset CAMAC

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE

Description

FUNCTION Send RESET CAMAC command .

File name I$ACV_RES_CAM.PPL

Action rout. I$ACV_RES_CAM

Dataset -

Version 1.01

Author H.G.Essel

Last Update 16-feb-1989

RESET MBD

RESET MBD

PURPOSE	Reset MBD and release all active channels
PARAMETERS	
FUNCTION	Reset the MBD and release all active channels. All current operations of the MBD are stopped and the active VAX I/O channels using the MBD are aborted and thereby released, i.e. no VMS MWAIT condition should occure.
REMARKS	The command should only be used if the MBD hangs.
EXAMPLE	RESET MBD
Action rout.	I$MCRSM
Author	M. Richter

Remarks

File name	I$MCRSM.PPL
Dataset	-

Description

CALLING	@CALL I$MCRSM;
ARGUMENTS	
FUNCTION	Reset the MBD and release all active channels. All current operations of the MBD are stopped and the active VAX I/O channels using the MBD are aborted and thereby released, i.e. no VMS MWAIT condition should occure.
REMARKS	-
EXAMPLE	@CALL I$MCRSM();

SEND DATA

SEND DATA VMEcrate,processor ID dummy crate node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Read one subevent.

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

crate Crate number

node NET node

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE SEND DATA

Description

FUNCTION Read one subevent.

File name I$ACV_SEND_DATA.PPL

Action rout. I$ACV_SEND_DATA

Dataset -

Version 1.01

Author H.G.Essel

Last Update 19-apr-1991

SET ACQUISITION

```
SET ACQUISITION in_buffers out_buffers events
                /[NO]SYNCHRONOUS
                /[NO]EXCLUSIVE
                /MAILBOX /NET
                /[NO]CHECK
                /[NO]COMPRESS
                /[NO]KEEP
                /[NO]START
                /[NO]STOP
```

PURPOSE Set data taking parameters.

PARAMETERS

in_buffers integer default=4
Number of buffers queued to MBD.
Default should be adequate.

out_buffers integer default=0
Number of buffers hold before writing to the output device is started.
Default should be adequate.

events integer replace default=0
Number of events to be filled into one buffer.
Zero means maximum (J11 only).

/SYNCHRONOUS Sychronous mode is entered. This means the Transport manager waits for the analysis. All buffers accepted by the transport manager are analyzed. The analysis may get the data via the first mailbox channel or via DECnet channel. The other channels are filled too, but not used for synchronization.

/EXCLUSIVE Enter exclusive mode. One buffer is sent only to one mailbox and one DECnet channel. Otherwise, a buffer may be sent to several analysis programs.

/MAILBOX Mailbox 1 is the controlling channel for synchronous mode (=default).

/NET	DECnet 1 is the controlling channel for synchronous mode.
/[NO] CHECK	Check buffer structure (default=/CHECK) /NOCHECK saves CPU time, but NO TYPE BUFFER or TYPE EVENT is possible.
/[NO] COMPRESS	J11 only: events are written in compressed mode. Default is /NO-COMPRESS. Compress mode is useful only if less then 30% of the event parameters are non zero. The buffers and events are marked so that the analysis can use the appropriate unpack routine.
/[NO] KEEP	J11 only: Keep data buffers (=default).
/[NO] START	(de)activate calling of start module. The module must be loaded first by LOAD MOD ACQ image module init /START
/[NO] STOP	(de)activate calling of stop module. The module must be loaded first by LOAD MOD ACQ image module init /STOP The modules must be linked into a sharable image by DCL command LSHARIM.
EXAMPLE	SET ACQU IN_BUF=6

Description

FUNCTION	Changes acquisition parameters initially defined by INIT ACQUIS command.
File name	I$ACQ_SET_ACQ.PPL
Action rout.	I$ACQ_SET_ACQ
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

SET ANALYSIS

SET ANALYSIS
/[NO]ANALYSIS
/[NO]DYNAMIC
/[NO]SYNCHRON
/[NO]EVENT
/[NO]START
/[NO]STOP
/[NO]FOREIGN
/[NO]TABLES

PURPOSE Set analysis parameters.

PARAMETERS

/[NO] ANALYSIS Switch ON/OFF calling of analysis routine in the event loop

/[NO] DYNAMIC Switch ON/OFF calling of dynamic list executor in the event loop

/[NO] SYNCHRON Switch ON/OFF DECnet output synchronization

/[NO] EVENT Select event or buffer unpacking.

/[NO] START (de)activate calling of start module. The module must be loaded first by
LOAD MOD ANAL image module init /START

/[NO] STOP (de)activate calling of stop module. The module must be loaded first by
LOAD MOD ANAL image module init /STOP

/[NO] FOREIGN Call foreign unpack routine X$UPFOR. The data buffer may have different structure than GOOSY buffers. Buffers are not checked in FOREIGN mode.

/[NO] TABLES Clear spectrum execution tables

REMARKS -

Description

FUNCTION This command allows to set analysis parameters:

1. Switch ON/OFF calling X$ANAL in event loop.
2. Switch ON/OFF calling dyn.list.exec.
3. Switch ON/OFF DECnet output synchronization.
4. Select buffer or event unpacking.
5. Enable/disable calling of start module
6. Enable/disable calling of stop module
7. Enable/disable calling of X$UPFOR.

File name I$ANACM.PPL

Action rout. I$ANACM_SET

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

SET DYNAMIC LIST

SET DYNAMIC LIST dyn_list dyn_type key value dyn_dir base node

PURPOSE Modify attached dynamic list

PARAMETERS

dyn_list Dynamic list name specification
If *, all attached lists are modified.
common required default

dyn_type Type of dynamic sublist or *
default:'*'

key Keyword for parameter to be changed.
LIST value=ON/OFF
SUBLIST value=ON/OFF

value Value for parameter

dyn_dir Default directory
common default:'$DYNAMIC'

base Default data base name
common default:'DB'

node Default node name
common default:'E'

EXAMPLE -

Caller M$DLCMD

Author H.G. Essel

File name M$ASTLL.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$ASTLL(CV_DYN_LIST,CV_TYPE,CV_PARAM,
CV_VALUE,CV_DYN_DIR,CV_BASE,CV_NODE)

ARGUMENTS

CV_DYN_LIST Dynamic list name specification
CHAR(*) VAR
Input

CV_TYPE Type of sublist or *
CHAR(*) VAR
Input

CV_PARAM Parameter name
CHAR(*) VAR
Input
List of parameters and values:
LIST ON/OFF
SUBLIST ON/OFF

CV_VALUE Value for parameter
CHAR(*) VAR
Input

CV_DYN_DIR Default Directory
CHAR(*) VAR
Input

CV_BASE Default Data Base name
CHAR(*) VAR
Input

CV_NODE Default node name
CHAR(*) VAR
Input

FUNCTION Set parameters of dynamic list

REMARKS Module is an action routine.

EXAMPLE -

SET EVENT INPUT

SET EVENT INPUT name type directory base

PURPOSE Set input event data element.

PARAMETERS

name string replace default=DB:[DATA]EVENT
DE name specification base:[dir]name

type string replace
Type of DE. If specified, this type is verified.
If the data element in the base has a different
type an error message is returned.

directory string replace default=DATA
Default directory, if not specified in name.

base string replace default=DB
Default base, if not specified in name.

FUNCTION All initialization entries of unpack routines (the loaded ones too) are called with the pointer to the data element and the length as arguments.

EXAMPLE SET EV IN DB:[DATA]MYEVENT

Description

FUNCTION This procedure locates the specified data element and calls $XEVENT, $XUPJ11, $XUPEVT and $XUPCMP.

File name I$ANACM.PPL

Action rout. I$ANACM_SET_EVI

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

SET EVENT OUTPUT

SET EVENT OUTPUT name type directory base

PURPOSE Set output event data element.

PARAMETERS

name string replace default=DB:[DATA]NEWEVENT
DE name specification base:[dir]name

type string replace
Type of DE. If specified, this type is verified.
If the data element in the base has a different
type an error message is returned.

directory string replace default=DATA
Default directory, if not specified in name.

base string replace default=DB
Default base, if not specified in name.

FUNCTION All initialization entries of pack routines (the loaded ones too) are called with the pointer to the data element and the length as arguments.

EXAMPLE SET EV OUT DB:[NEW]EVENT

Description

FUNCTION This procedure locates the specified data element and calls $XPACMP and $XPAEVT. Then the routines X$PACMP or X$PAEVT copy this data element into the output buffers. The routione is selected by START ANAL OUTPUT /COMPRESS or /COPY

File name I$ANACM.PPL

Action rout. I$ANACM_SET_EVO

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

SET FASTBUS PEDESTAL

SET FASTBUS PEDESTAL sample trigger
VMEcrate,processor ID dummy crate node
/ON/OFF [=ONOFF]
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Set fastbus pedestal subtraction on/off.

PARAMETERS

sample integer (def=100)
Sample interval [events]. After "sample" events one event will be not compressed.

trigger integer (def=1)
Trigger number

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

crate Crate number

node NET node

/ON/OFF Switch compression ON or OFF

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE SET VME TRIG

Description

FUNCTION	Set fastbus pedestal subtraction on/off.
File name	I$ACV_SET_PED.PPL
Action rout.	I$ACV_SET_PED
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

SET RANDOM

SET RANDOM type subtype channels datawords
/COMPRESS
/PRINT
/SPAN

PURPOSE Set some random generator parameters

PARAMETERS

type integer default=4
Data header type

subtype integer default=1
Data header subtype

channels integer default=4
channels per event

datawords integer default=4000
number of data words used in buffer

/COMPRESS switch
Generate compressed events

/PRINT switch
Output events

/SPAN swicth
allow buffer spanning

Description

PURPOSE Setup random generator of TMR.

SET SCATTER BUFFER

SET SCATTER BUFFER value dyn_list dyn_dir base node

PURPOSE Set scatter buffer size

PARAMETERS

value Number of display points to be collected, before the buffer is sent to display
default:'1000'

dyn_list Dynamic list name specification
common required default
If *, all attached lists are modified.

dyn_dir Default directory
common default:'$DYNAMIC'

base Default data base name
common default:'DB'

node Default node name
common default:'E'

EXAMPLE -

Caller M$DLCMD

Author H.G.Essel

File name M$ASTSB.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$ASTSB(CV_VALUE,CV_DYN_LIST
,CV_DYN_DIR,CV_BASE,CV_NODE)

ARGUMENTS

CV_VALUE Number of scatter points to be collected in buffer.

CV_DYN_LIST Dynamic list name specification

CV_DYN_DIR Default directory

CV_BASE Default data base name

CV_NODE Default node name

FUNCTION Set buffer size of scatter buffer. The buffer size is specified as number of scatter points to be stored in the buffer. Note that there is at least one point per scatter frame. If there are bit variables, several points per frame might be sent. The buffer size must not be set below the maximum number of points for a picture.

REMARKS Module is an action routine.

EXAMPLE -

SET VME BUFFER

SET VME BUFFER buffers size VMEcrate,processor ID
dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]
/STOP/RESET [=LAST]

PURPOSE Setup frontend buffers

PARAMETERS

buffers Number of buffers.

size Size of buffers (bytes)

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

/STOP/RESET Break stopping status. Using VME frontends, the STOP ACQ command will enter stopping status which terminated by last buffer. When the frontend system does not send buffers stopping status would never be terminated. In this case, /STOP or /RESET can be used to terminate this status.

EXAMPLE SET VME BUF 16 ID=20
SET VME BUF 0 16000 ID=20
Buffers are allocated in the free memory of the
frontend processor. When specifying the number of buffers, the size will be calculated. When specifying the size, the number will be calculated. One of the two, buffers or size, must be zero (default).

Description

FUNCTION Setup frontend buffers.

File name I$ACV_SET_VME_BUF.PPL

Action rout. I$ACV_SET_VME_BUF

Dataset -

Version 1.01

Author H.G.Essel

Last Update 16-feb-1989

SET VME CONTROL

SET VME CONTROL name value VMEcrate,processor ID
dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Set value in control structure

PARAMETERS

name Name of parameter as seen in GOOVME(SS$VMECTRL).

FIC_DEBUG 0 no output
1 informational output
2 debug output

value Integer value.

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE SET VME CONT FIC_DEBUG 1 /ALL

Description

FUNCTION	Set value in control structure
File name	I$ACV_SET_VME_CTRL.PPL
Action rout.	I$ACV_SET_VME_CTRL
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

SET VME INPUT

SET VME INPUT
/HVR /NET /TDAS /OFF /CHECK [=PATH]

PURPOSE	Select lmd data path
PARAMETERS	
/HVR	Use HVR interface.
/NET	Use NET interface.
/TDAS	Send buffers to TDAS event builder.
/OFF	Switch off
/CHECK	Check buffer structure only.
EXAMPLE	SET VME INP /HVR

Description

FUNCTION	Select lmd data path.
File name	I$ACV_SET_VME_INP.PPL
Action rout.	I$ACV_SET_VME_INP
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

SET VME TRIGGER

```
SET VME TRIGGER VMEcrate,processor ID dummy
                crate fastclear conversion node
                /RESET
                /MASTER
                /ENABLE/DISABLE [=ENABLE]
                /[NO]LOAD
                /ALL/FEP/EB [=DESTINATION]
                /CVI/CAV/EBI [=CONTROL]
```

PURPOSE Set trigger module.

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

crate Crate number

fastclear Time for fast clear

conversion Conversion time

node NET node

/RESET Reset trigger module

/MASTER Set master

/ENABLE Enable trigger

/DISABLE Disable trigger

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI	Select processor by controller
/[NO] LOAD	Do [NOT]execute. Default= /LOAD

EXAMPLE SET VME TRIG

Description

FUNCTION	Set trigger module.
File name	I$ACV_SET_TRIG.PPL
Action rout.	I$ACV_SET_TRIG
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

SHOW ACQUISITION

SHOW ACQUISITION timer output
/PRINT
/OUTFILE
/INFILE
/CLEAR
/RUN
/SETUP
/BRIEF
/[NO]RATE

PURPOSE	Show acquisition status
PARAMETERS	
timer	integer optional time intervall [sec]for /RATE default is 5 sec.
output	string optional output file
/PRINT	Print output
/CLEAR	Clear counter
/RUN	Show run information
/SETUP	Show VME setup information
/OUTFILE	Show current output file header
/INFILE	Show current input file header
/BRIEF	Brief output
/[NO] RATE	Show data rate
EXAMPLE	SHO ACQU

Description

FUNCTION	Acquisition parameters are output.
File name	I$ACQ_SHO_ACQ.PPL
Action rout.	I$ACQ_SHO_ACQ
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

SHOW ANALYSIS

SHOW ANALYSIS timer output
/PRINT
/BRIEF
/OUTFILE
/INFILE
/CLEAR
/[NO]RATE

PURPOSE	Show analysis status
timer	integer optional time intervall [sec]for /RATE default is 5 sec.
output	string optional output file
/PRINT	print output
/BRIEF	Brief output
/CLEAR	clear counters
/OUTFILE	Show current output file header
/INFILE	Show current input file header
/[NO] RATE	Show data rate
PURPOSE	Show analysis status
EXAMPLE	SHOW ANA

Description

FUNCTION	This procedure writes some information about the present analysis status to terminal
File name	I$ANACM.PPL

Action rout.	I$ANACM_SHOW
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

SHOW DYNAMIC ATTACHED

SHOW DYNAMIC ATTACHED dyn_list dyn_type dyn_dir base node output
/PRINT
/[NO]QUEUE
/FULL

PURPOSE Show attached dynamic list

PARAMETERS

dyn_list Dynamic list name specification
required
common default
If *, all attached lists are shown.

dyn_type Type of dynamic sublist or *
default:'*'

dyn_dir Default directory
default:'$DYNAMIC'
common default

base Default data base name
default:'DB'
common default

node Default node name
default:'E'
common default

output Optional output file

/PRINT Print output

/[NO] QUEUE Display queue content

/FULL Display full queue content

EXAMPLE -

Caller M$DLCMD

Author H.G.Essel

File name M$ASHLL.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING

```
STS=M$ASHLL(CV_DYN_LIST,CV_TYPE,
            CV_DYN_DIR,CV_BASE,CV_NODE,CV_OUT,
            I_PRINT,I_QUEUE,I_FULL)
```

ARGUMENTS

CV_DYN_LIST Dynamic list name specification
CHAR(*) VAR
Input

CV_TYPE Type of sublist or *
CHAR(*) VAR
Input

CV_DYN_DIR Default Directory
CHAR(*) VAR
Input

CV_BASE Default Data Base name
CHAR(*) VAR
Input

CV_NODE Default node name
CHAR(*) VAR
Input

CV_OUT Optional output file
CHAR(*) VAR
Input

I_PRINT	If 1, print output
I_QUEUE	If 1, the content of the queues is displayed.
I_FULL	If 1, the content of the queues is fully displayed.
FUNCTION	Show local attached dynamic list If dynamic list name is *, all attached lists are shown.
REMARKS	Module is an action routine.
EXAMPLE	-

SHOW SCATTER BUFFER

SHOW SCATTER BUFFER dyn_list dyn_dir base node

PURPOSE Show scatter buffer size

PARAMETERS

dyn_list Dynamic list name specification
common required default
If *, all attached lists are shown.

dyn_dir Default directory
common default:'$DYNAMIC'

base Default data base name
common default:'DB'

node Default node name
common default:'E'

EXAMPLE -

Caller M$DLCMD

Author H.G.Essel

File name M$ASHSB.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$ASHSB(CV_DYN_LIST
,CV_DYN_DIR,CV_BASE,CV_NODE)

ARGUMENTS

CV_DYN_LIST Dynamic list name specification

CV_DYN_DIR Default directory

CV_BASE Default data base name

CV_NODE Default node name

FUNCTION Show buffer size of scatter buffer. The buffer size is specified as number of scatter points to be stored in the buffer. Note that there is at least one point per scatter frame. If there are bit variables, several points per frame might be sent. The buffer size must not be set below the maximum number of points for a picture.

REMARKS Module is an action routine.

EXAMPLE -

SHOW STARBURST

SHOW STARBURST C=c N=n

PURPOSE Show the execution parameters of a STARBURST.

PARAMETERS

C=c Crate number of the STARBURST to be accessed. The default is 1.

N=n Crate number of the STARBURST to be accessed. The default is 23.

FUNCTION This command displays the execution parameters of a STARBURST processor running the simple data aquisition executive.
The command reads the vector and communication area located between %O600 and %O700. The first 8 words are used for status variables and pointers shared between MBD and STARBURST:

- %O600 : New status
- %O602 : Old status
- %O604 : Address of a valid event buffer
- %O606 : Subsystem index
- %O610 : Address of the first parameters set
- %O612 : Length of the first parameter set.

The procedure displays the current status, prints a warning if the subsystem index differs from the crate number, prints the last event buffer if there is a valid one and lists the declared parameter sets.

EXAMPLE DUMP STARBURST C=1 N=23

Action rout. I$MCSHS

Author Walter F.J. Mueller

Remarks

File name I$MCSHS.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$MCSHS(I_C,I_N);

ARGUMENTS

I_C BIN FIXED(15) [INPUT]
Crate number of STARBURST to be accessed.

I_N BIN FIXED(15) [INPUT]
Station number of STARBURST to be accessed.

FUNCTION Show the execution parameters of a STARBURST. It dump some locations in the vector page. For details look in the command description.

REMARKS -

EXAMPLE @CALL I$MCSHS(1,23);

SHOW VME CONTROL

SHOW VME CONTROL name VMEcrate,processor ID
dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Show values in control structure

PARAMETERS

name Name of parameter as seen in GOOVME(SS$VMECTRL). If not specified, the whole structure is displayed.

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE SET VME CONT FIC_DEBUG 1 /ALL

Description

FUNCTION Show values in control structure

File name I$ACV_SHO_VME_CTRL.PPL

Action rout. I$ACV_SHO_VME_CTRL

Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	16-feb-1989

SHOW VME SETUP

SHOW VME SETUP file /FULL

PURPOSE	Show VME setup.
PARAMETERS	
file	Optional output file
/FULL	Show full information.
EXAMPLE	SHO VME SET /F

Description

FUNCTION	-
File name	I$ACVME.PPL
Action rout.	I$ACVME_SHO_VME
Dataset	-
Version	1.01
Author	H.G.Essel
Last Update	20-Jul-1989

START ACQUISITION

START ACQUISITION buffers events
skip_buf skip_event
/CLEAR /NET /STOP /RESET

PURPOSE Start data taking

PARAMETERS

buffers integer default=0
Number of buffers to process (0 = infinite)
(file input only)

events integer default=0
Number of events to process (0 = infinite)
(file input only)

skip_buf integer default=0
Number of buffers to skip
(file input only)

skip_event integer default=0
Number of events to skip
(file input only)

/NET switch
Start input from net (VME)

/CLEAR switch
Clear counter. Same effect as SHOW/CLEAR.
Some counters are cleared by START ACQUISITION, others by SHOW/CLEAR or START ACQ/CLEAR.

/STOP/RESET switch
Break stopping status. Using VME frontends, the STOP ACQ command will enter stopping status which terminated by last buffer. When the frontend system does not send buffers stopping status would never be terminated. In this case, /STOP or /RESET can be used to terminate this status.

EXAMPLE STA ACQU

Description

FUNCTION	This procedure starts the data acquisition. The MBD and all front end processors are started. MBD is initialized by I$ACQ_IMBD. Input is queued by delivering I$ACQ_QMBD on AST level. For a J11 system the J11 is started. The buffer and event counters are cleared.
File name	I$ACQ_STA_ACQ.PPL
Action rout.	I$ACQ_STA_ACQ
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

START ANALYSIS OUTPUT

START ANALYSIS OUTPUT file size buffersize
device directory
type subtype stream
headerinput headeroutput
/PROMPT
/EDIT
/[NO]OPEN
/[NO]SYNCHRON
/COPY/COMPRESS/INPUT
/MBD/J11
/BYTE/KBYTE/PAGE/BUFFER

PURPOSE

Start data output from analysis. Output is done to DECnet. If a file is specified, output is written to file too.

PARAMETERS

file
string
File specification, disk or tape.
Keep in mind filename comventions for IBM.

size
integer default=35000
maximal file size in Kbytes.

buffersize
integer default=16384
Buffer size in bytes.

device
string replace
Default device

directory
string replace
Default directory

type
integer replace
Type for output buffer (default=4)
Ignored if /MBD or /J11 is specified.

subtype	integer replace Subtype for output buffer (default=1) Ignored if /MBD or /J11 is specified.
stream	integer replace Maximum numbers of buffers containing event fragments
headerinput	string replace default=" Optional file to read file header.
headeroutput	string replace default=" Optional file to store file header.
/PROMPT	Prompt file header information
/EDIT	Edit file header (headerinput required).
/OPEN	Open new output file (=default)
/NOOPEN	Continue writing to old file. Only possible if files output has been stopped by STOP OUT FILE /NOCLOSE
/SYNCHRON	Set synchron mode. Wait for output to DECnet, if a DECnet channel has been opened by another anaysis.
/MBD	Use buffer type/subtype like MBD buffers.
/J11	Use buffer type/subtype like J11 buffers. In both cases the type subtype parameters described above are ignored.
/COPY	Use X$PAEVT for copying output event into output buffer (=default).
/INPUT	Use X$PAEVT for copying event into output buffer. The difference to /COPY is that the original event from input buffer is copied. No output data element is needed or used.
/COMPRESS	Use X$PACMP for packing Output event into output buffer.
/PAGE	file size in pages
/BYTE	file size in bytes
/KBYTE	file size in Kbyte
/BUFFER	file size in buffers

EXAMPLE

START ANAL OUT DAY$ROOT:[SCRATCH]F1.LMD
copies output data element to output buffer with
type/subtype 4,1.
SET ANAL/EVENT
STA ANA OUT X /MBD/INPUT
copy MBD events directly from input buffer to
output buffer. Event unpacking must be enabled!
STA AN OUT X /COMP
copy compressed output data element to buffer.
In the buffer the event wil have a standard type.
When these events are read by analysis, they are
decompressed and copied into the data element.
If one wants to send the output files to the IBM, the filenames must follow some conventions:

Maximal length 25 char (including type)
Maximal 8 char or 7 digits between two _
File type is .LMD

Description

FUNCTION

This procedure starts data output from analysis. Output is done to DECnet. If a file is specified, output is written to file too. The AMR checks if another analysis program has opened a link. In this case the output buffer is written to this DECnet channel. If /SYNCHRON is set, it waits for the acknowledge. Otherwise buffers are written to the DECnet channel only if the receiver acknowledged the previous buffer.
If one wants to send the output files to the IBM, the filenames must follow some conventions:

Maximal length 25 char (including type)
Maximal 8 char or 7 digits between two _
File type is .LMD

File name I$ANACM.PPL

Action rout. I$ANACM_STA_OUT

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

START ANALYSIS RANDOM

START ANALYSIS RANDOM bufevents events

PURPOSE Start analysis for Monte Carlo.

PARAMETERS

bufevents integer default=100
Number of events per virtual buffer to process.

events integer default=0
Number of events to process (0 = infinite)

EXAMPLE START ANAL RAN 100

Description

FUNCTION This command starts the analysis without input. No event is copied into the data base. The user analysis routine must generate its own raw data. The second argument specifies, how many events (Calls of X$ANAL) are executed before a command can be executed. Note, that X$ANAL may return status XIO_STOPINPUT to stop the analysis.

File name I$ANACM.PPL

Action rout. I$ANACM_STA_RAN

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

START DYNAMIC LIST

START DYNAMIC LIST
dyn_list dyn_type dyn_dir base node

PURPOSE Start execution of dynamic list

PARAMETERS

dyn_list Dynamic list name specification
required
common default
If *, all attached lists are modified.

dyn_type Type of dynamic sublist or *
Empty : Start main list
* : Start all sublists
type : Start sublist "type"
default:"

dyn_dir Default directory
default:'$DYNAMIC'
common default

base Default data base name
default:'DB'
common default

node Default node name
default:'E'
common default

EXAMPLE -

Caller M$DLCMD

Author H.G.Essel

File name M$ASTDL.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$ASTDL(CV_DYN_LIST,CV_TYPE
,CV_DYN_DIR,CV_BASE,CV_NODE)

ARGUMENTS

CV_DYN_LIST Dynamic list name specification

CV_TYPE Type of sublist
Empty : Start main list
* : Start all sublists
type : Start sublist "type"

CV_DYN_DIR Default directory

CV_BASE Default data base name

CV_NODE Default node name

FUNCTION Start execution of dynamic list

REMARKS Module is an action routine.

EXAMPLE -

START INPUT FILE

```
START INPUT FILE file buffers events
                 skip_buffer skip_event
                 device directory
                /CLEAR
                /OPEN
                /FOREIGN
                /[NO]HEADER
```

PURPOSE Start data analysis from file at current position. Open it if it was not open.

PARAMETERS

file required string replace
File specification, disk or tape

buffers integer default=0
Number of buffers to process (0 = infinite)
When this number of buffers is processed, input stops, but the file remains open. The next START INPUT FILE command continues. Skipped buffers are not counted.

events integer default=0
Number of events to process (0 = infinite)
When this number of events is processed, input stops, but the file remains open. The next START INPUT FILE command continues. Events skipped by command are not counted.

skip_buffer integer default=0
Number of buffers (records) to skip

skip_event integer default=0
Number of events to skip

device string replace
Default device

directory string replace
Default directory

/CLEAR	Clear counters, even if file is continued.
/OPEN	Open new file. Close it first, if it was open.
/FOREIGN	Foreign buffer format. I$buffer calls unpack routine X$UPFOR
/HEADER	File has GOOSY header (=default).
/NOHEADER	File has no GOOSY header. If the file has no header, but /HEADER is specified, the file must be closed after reading the first buffer and opened again. This can be time consuming on tapes.
EXAMPLE	START INP FILE DAY$ROOT:[SCRATCH]F1.LMD START INP FILE M1:F1.LMD 1000
NOTE	It is recommended to skip/process either events or buffers, because events in skipped buffers are not counted. When processing a number of events, there are two buffers pending after stop. The one with the last event, and the next one. These two buffers are processed first with next START INPUT FILE command, except /OPEN was given. Skipping two buffers these two buffers are skipped. Otherwise processing starts with the next event in the first pending buffer.

Description

FUNCTION	This procedure starts data taking from file at current position. If no file is open, it will be opened in any case. If a file is open, it will be closed and reopened, if /OPEN is specified, but will remain open at the same position, if /OPEN is NOT specified. When an optional buffer/event limit is reached, file input stops, but file remains open.
File name	I$ANACM.PPL
Action rout.	I$ANACM_STA_FIL
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

START INPUT MAILBOX

START INPUT MAILBOX mbx_name mbx_number
buffers events bufevents
skip_buffers size

PURPOSE Open input stream from mailbox

PARAMETERS

mbx_name string replace
name of mailbox GOOSY_name_# (def.=environment)

mbx_number integer replace default=1
Number of mailbox GOOSY_name_# (1,2,3) (def.=1)

buffers integer default=0
Number of buffers to process (0 = infinite)

events integer default=0
Number of events to process (0 = infinite)

bufevents integer default=0
Number of events per buffer to process (0=infinite)

skip_buffers integer default=0
Number of buffers to be skipped

size integer replace default=8192
Buffersize in bytes. This size must match the size
as specified in INI ACQUIS, i.e. 8192 for MBD (=default) and 8192 for J11 single crate system.

EXAMPLE START INPUT MAILBOX SUSI SIZE=8192

Description

FUNCTION This procedure opens a mailbox to read data from TMR. The name of the mailbox is GOOSY_name_n. The mailbox must exist (is created by TMR).

File name	I$ANACM.PPL
Action rout.	I$ANACM_OP_MBX
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

START INPUT NET

START INPUT NET node environment component
buffers events
/TMR/ANL
/MULTI

PURPOSE Open input stream from network

PARAMETERS

node required string replace
Name of node, where partner runs

Environment required string replace
Name of the environment of the partner

Component string default=$TMR
Name of the component of the partner (default=$TMR)

buffers integer default=0
Number of buffers to process (0 = infinite)

events integer default=0
Number of events to process (0 = infinite)

/TMR Net input from transport manager (=default)

/ANL Net input from analysis

/MULTI Allow multiple input links

EXAMPLE
```
START INPUT NET B TEST $TMR
START INPUT NET B TEST $ANL /ANL
```

Description

FUNCTION This procedure opens a link to TMR or ANL processes to get data.

File name I$ANACM.PPL

Action rout. I$ANACM_OP_NET

Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

START OUTPUT FILE

```
START OUTPUT FILE file size number
                  device directory
                  headerinput headeroutput
                 /PROMPT
                 /EDIT
                 /[NO]OPEN
                 /AUTOMATIC
                 /ALLOCATE
                 /PAGE /BYTE /KBYTE /BUFFER
```

PURPOSE Start list mode dump

PARAMETERS -

file required string replace
File name for new file, if required.
Keep in mind filename comventions for IBM.

size integer replace default=35000
Size of new file in Kbytes, if required
Units can be selected by /PAGE/BYTE/BUFFER.

number integer replace default=1000
Number of files written automatically.
(used for /AUTO)

device string replace
Default device

directory string replace
Default directory

headerinput string replace default="
Optional file to read file header.

headeroutput string replace default="
Optional file to store file header.

/PROMPT Prompt file header information

/EDIT	Edit file header (headerinput required).
/NOOPEN	continue output in current file This can be done only if a previous STOP OUT FILE /NOCLOSE was given. Once a file is closed, it cannot be continued.
/OPEN	open new file (close current) (default)
/AUTOMATIC	A new file is opened, if the previous one is filled. A three digit current number is appended to the filename. The /OPEN switch is required.
/ALLOCATE	Preallocate file (disk only)
/PAGE	File size in pages (512 bytes)
/BYTE	File size in bytes
/KBYTE	File size in Kbytes (1024 bytes =default)
/BUFFER	File size in buffers
EXAMPLE	STA OUT FIL /NOCLOSE (continue current open file) STA OUT FIL x.lmd (start new file) If one wants to send the output files to the IBM, the filenames must follow some conventions: Maximal length 25 char (including type) Maximal 8 char or 7 digits between two underscore No dollar signs. File type is .LMD

Description

FUNCTION	Start list mode dump to file.
File name	I$ACQ_STA_LMD.PPL
Action rout.	I$ACQ_STA_LMD
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

START RUN

START RUN name

PURPOSE Start run.

PARAMETERS

name Run name or @filename containing run information. The file is supposed to be ASCII, 80 char/line.

EXAMPLE STA RUN @RUN_AU_173.RUN

Description

FUNCTION This procedure starts a run. The run name or filename is stored in the control structure. All frontend processors get the command. With the SHOQ ACQUIS command the content of the run file may be displayed.

File name I$ACQ_STA_RUN.PPL

Action rout. I$ACQ_STA_RUN

Version 1.01

Author H.G.Essel

Last Update 4-Sep-1991

START VME

START VME VMEcrate,processor ID dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Send START command to NET

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE START VME 1,1

Description

FUNCTION Send START command .

File name I$ACV_STA_VME.PPL

Action rout. I$ACV_STA_VME

Dataset -

Version 1.01

Author H.G.Essel

Last Update 16-feb-1989

STOP ACQUISITION

STOP ACQUISITION /ABORT /CLOSE /STOP/RESET

PURPOSE	Stop data taking
PARAMETERS	-
/ABORT	Delete links to J11 and cleanup (J11 only). All buffers in the J11 are lost.
/CLOSE	Close input file (File input only)
/STOP/RESET	Do not wait for last buffer.
EXAMPLE	STOP ACQU
FUNCTION	With the MBD as frontend all data in the MBD is sent to the VAX. All data is written to file, if output is active. With a single crate J11 system the /ABORT qualifier clears all data in the J11.

Description

FUNCTION	MBD inputs are canceled. J11 is stopped. Mailbox QIO's are canceled.
File name	I$ACQ_STO_ACQ.PPL
Action rout.	I$ACQ_STO_ACQ
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

STOP ANALYSIS OUTPUT

STOP ANALYSIS OUTPUT /[NO]CLOSE

PURPOSE	Stop data output from analysis
PARAMETERS	
/CLOSE	Close output file (=default).
EXAMPLE	STOP ANAL OUT

Description

FUNCTION	This procedure stops data output from analysis.
File name	I$ANACM.PPL
Action rout.	I$ANACM_STO_OUT
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

STOP ANALYSIS RANDOM

STOP ANALYSIS RANDOM

PURPOSE	Close input stream from mailbox
PARAMETERS	
EXAMPLE	STOP ANAL RAN

Description

FUNCTION	This procedure stops the analysis.
File name	I$ANACM.PPL
Action rout.	I$ANACM_STO_RAN
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

STOP DYNAMIC LIST

STOP DYNAMIC LIST dyn_list dyn_type dyn_dir base node

PURPOSE Stop execution of dynamic list

PARAMETERS

dyn_list Dynamic list name specification
If *, all attached lists are modified.
common required default

dyn_type Type of dynamic sublist or *
Empty : Start main list
* : Start all sublists
type : Start sublist "type"
default:"

dyn_dir Default directory
common default:'$DYNAMIC'

base Default data base name
common default:'DB'

node Default node name
common default:'E'

EXAMPLE -

Caller M$DLCMD

Author H.G.Essel

File name M$ASPDL.PPL

Dataset -

Remarks

REMARKS -

Description

CALLING STS=M$ASPDL(CV_DYN_LIST,CV_TYPE,CV_DYN_DIR, CV_BASE,CV_NODE)

ARGUMENTS

CV_DYN_LIST Dynamic list name specification

CV_TYPE Type of sublist
Empty : Start main list
* : Start all sublists
type : Start sublist "type"

CV_DYN_DIR Default directory

CV_BASE Default data base name

CV_NODE Default node name

FUNCTION Stop execution of dynamic list

REMARKS Module is an action routine.

EXAMPLE -

STOP INPUT FILE

STOP INPUT FILE
/CLOSE

PURPOSE Stop reading input file, optional close.

PARAMETERS

/CLOSE close file. File will be opened again by start with START INP FILE. Otherwise keep file open. Continue with START INP FIL at the same position.

EXAMPLE STOP INPUT FILE

Description

FUNCTION This procedure stops reading input file. The input stream may be resumed by START INPUT FILE command at the same position, if stopped without /CLOSE

File name I$ANACM.PPL

Action rout. I$ANACM_STO_FIL

Version 1.01

Author H.G.Essel

Last Update 12-APR-1985

STOP INPUT MAILBOX

STOP INPUT MAILBOX mbx_num

PURPOSE	Close input stream from mailbox
PARAMETERS	
mbx_num	integer replace default=1 Number of mailbox (1,2,3). The mailbox' name is GOOSY_name_n, n=1,2,3
EXAMPLE	STOP INPUT MAIL 1

Description

FUNCTION	This procedure closes a mailbox to read data from TMR.
File name	I$ANACM.PPL
Action rout.	I$ANACM_CLO_MBX
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

STOP INPUT NET

STOP INPUT NET

PURPOSE	Close input stream from DECnet
EXAMPLE	STOP INPUT NET

Description

FUNCTION	This procedure closes a link to read data from TMR.
File name	I$ANACM.PPL
Action rout.	I$ANACM_CLO_NET
Version	1.01
Author	H.G.Essel
Last Update	12-APR-1985

STOP OUTPUT FILE

STOP OUTPUT FILE
/[NO]CLOSE

PURPOSE	Stop list mode dump
PARAMETERS	-
/NOCLOSE	Keep file open to continue with STA OUT FILE /NOOP
/CLOSE	Close file (NO append possible) (default)
EXAMPLE	STOP OUT FILE (file is closed)

Description

FUNCTION	Stop list mode dump to file. Optional the file is closed.
File name	I$ACQ_STO_LMD.PPL
Action rout.	I$ACQ_STO_LMD
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

STOP RUN

STOP RUN /STOP /ABORT /CLOSE

PURPOSE	Stop run.
PARAMETERS	-
/STOP	Stop acquisition first.
/ABORT	With /STOP: Delete links to J11 and cleanup (J11). All buffers in the J11 are lost.
/CLOSE	With /STOP: Close input file (File input only)
EXAMPLE	STOP RUN
FUNCTION	Sends STOP RUN command to frontend systems (VME). When /STOP is given, stops acquisition first and closes output file. The /ABORT and /CLOSE switches are used only with /STOP

Description

FUNCTION	Stops run. Optionally stop acquisition first.
File name	I$ACQ_STO_RUN.PPL
Action rout.	I$ACQ_STO_RUN
Version	1.01
Author	H.G.Essel
Last Update	4-Sep-1991

STOP VME

STOP VME VMEcrate,processor ID dummy node
/LOAD
/ALL/FEP/EB [=DESTINATION]
/CVI/CAV/EBI [=CONTROL]

PURPOSE Send STOP command to NET

PARAMETERS

VMEcrate,processor List of processor specifications, i.e. 1,0,1,1,1,2 for processors with offets 0,1,2 in VME crate 1

ID integer
Processor ID

dummy NOT used

node optional node name of NET

/ALL/FEP/EB Select processor

/CVI/CAV/EBI Select processor by controller

/[NO] LOAD Do [NOT]execute. Default= /LOAD

EXAMPLE STOP VME 1,1

Description

FUNCTION Send STOP command .

File name I$ACV_STO_VME.PPL

Action rout. I$ACV_STO_VME

Dataset -

Version 1.01

Author H.G.Essel

Last Update 16-feb-1989

STORE LRS_2365

STORE LRS_2365 C=c N=n FILE=file
/DUMP/NODUMP

PURPOSE Read back definitions from a LRS 2365 logic matrix and write them formated to a file (or SYS$OUTPUT).

PARAMETERS

C=c Crate number of the logic matrix to be accessed.

N=n Station number of the logic matrix to be accessed.

FILE=file Definition file to be written. The default file type is DAT. The format is as described for the LOAD LRS_2365 command. The default is SYS$OUTPUT.

/DUMP Dumps the contents of the LRS 2365 module found after the down load. The programming words PW 0 to 17 are shown in binary radix. For their interpretation look in the LRS manual.

/NODUMP Don't dump programming words, this is the default.

FUNCTION This commands reads the setup of a LRS 2365 logic matrix back and generates a definition file. This file is in the format described for the LOAD LRS_2365 command and may be used at a later time with this command.
This command also serves as a DUMP type command if the FILE parameter is omitted. The definition file is written to SYS$OUTPUT in this case.

EXAMPLE STORE LRS_2365 1 1

Action rout. I$STLRS_2365

Author Walter F.J. Mueller

Remarks

File name I$STLRS_2365.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$STLRS_2365(I_C,I_N,C_FILE,C_DUMP);

ARGUMENTS

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be loaded.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be loaded.

C_FILE CHAR(*) VAR [INPUT]
Name of definition file to be written.

C_DUMP CHAR(*) [INPUT]
DUMP qualifier set:

/DUMP	Dump programming words.
/NODUMP	Don't dump programming words.

FUNCTION Reads the setup of a LRS 2365 logic matrix and writes a formated definition file.

REMARKS -

EXAMPLE @CALL I$STLRS_2365(1,1,'SYS$OUTPUT',");

STORE MBD

STORE MBD file

PURPOSE Store a MBD memory dump in a file.

PARAMETERS

file Name of the dump file to be written. The default file type is .BDD. The file will contain 16 records with 512 bytes each with the binary image of the whole MBD memory. This file can be read and dumped with the DUMP MBD command.

FUNCTION This procedure writes a dump of the whole MBD memory to a file. This may be usefull after an MBD error or other unexpected malfunctions. The file can be read later with the DUMP MBD command and may allow to reconstuct the source of trubble.

EXAMPLE STORE MBD MYDUMP

Action rout. I$MCSTM

Author Walter F.J. Mueller

Remarks

File name I$MCSTM.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$MCSTM(C_FILE);

ARGUMENTS

C_FILE CHAR(*) VAR [INPUT]
Name of the dump file to be written. The default file type is .BDD.

FUNCTION This procedure reads the whole 4 kwords MBD memory and writes it in a dump file. This file has 16 records with 512 bytes each containing the binary memory image.

REMARKS -

EXAMPLE @CALL I$MCSTM('MYDUMP');

TEST BOR_1802

TEST BOR_1802
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP
/FULL

PURPOSE Perform tests with a BORER 1802 Dataway display

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

/FULL A full loopback test will be done in every test cycles. This is especially usefull for 'online' debugging of branch cables ect. because each bit of the CAMAC system is tested in all cycles thus providing a fast response for intermittend errors.

FUNCTION This test uses a Dataway display (either a BORER 1802 or compatible types) as a loopback mirror and tries to verify the datatransfer thru the branch driver, the branch cables, the crate controller the Dataway and back. This is done by sending test patterns to the Dataway display and reading them back. The full test algorithm works as follows:

Set DWD to online mode to aviod contentions with active auxiliary crate controllers.

Generate a new test pattern (for details of look in the description of I$GTPAT).

Send the test pattern to the DWD, read back, compare and check X-Q responses.

Set DWD back to monitor mode.
If the /FULL qualifier has been specified, a set of 532 test pattern is generated and processed in all test cycle. In this mode tests each bit in the CAMAC system in all cycles.

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

EXAMPLE TEST BOR_1802 C=1 N=22 /START

Action rout. I$CTBOR_1802

Author Walter F.J. Mueller

Remarks

File name I$CTBOR_1802.PPL

Dataset -

Description

CALLING

```
@CALL I$CTBOR_1802(I_B,I_C,I_N,F_DELTA,
                   C_OPTION,C_FULL);
```

ARGUMENTS

I_B	BIN FIXED(15) [INPUT] Branch number of the module to be tested.
I_C	BIN FIXED(15) [INPUT] Crate number of the module to be tested.
I_N	BIN FIXED(15) [INPUT] Station number of the module to be tested.
F_DELTA	BIN FLOAT(24) [INPUT] Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..
C_OPTION	CHAR(*) VAR [INPUT] Test option qualifier set, expected values are: /LIST/STOP /RUN/START/LOOP
C_FULL	CHAR(*) VAR [INPUT] /FULL qualifier. If specified, all test patters are written and read in every test interation.
FUNCTION	This procedure performs Dataway loopback tests with a BORER 1802 Dataway display as mirror. For a more detailed discussion look in the description of the command TEST BOR_1802.
REMARKS	-
EXAMPLE	@CALL I$CTBOR_1802(0,1,22,0.E0,'/RUN','/FULL');

TEST CAMAC

TEST CAMAC B=b C=c N=n TYPE=*
/STOP
/LIST

PURPOSE Common functions for CAMAC tests.

PARAMETERS

B=b Branch for which tests are to be affected. Tests in all branches are affected if omitted or specified as 0 (default = 0).

C=c Crate for which tests are to be affected. Tests in all crates are affected if omitted or specified as 0 (default = 0).

N=n Station number for which test are to be affected. Tests in all stations are affected if omitted or specified as 0 (default = 0).

TYPE=t Type of tests to be affected. Tests of all types are affected if omitted or specified as '*'.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

EXAMPLE
```
TEST CAMAC /LIST
  TEST CAMAC TYPE=BOR_1802 /STOP
```

REMARKS The command will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTCAMAC

Author Walter F.J. Mueller

Remarks

File name I$CTCAMAC.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$CTCAMAC(I_B,I_C,I_N,C_TYPE,C_OPTION);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

C_TYPE CHAR(*) VAR [INPUT]
Type of the tests to be selected, either a '*' or the name of test.

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP

FUNCTION This procedure performs some functions acting on all or a group of CAMAC tests. For a more detailed discussion look in the description of the command TEST CAMAC.

REMARKS -

EXAMPLE @CALL I$CTCAMAC(0,0,0,'*','/STOP');

TEST GSI_IOL

TEST GSI_IOL
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP
/LOOPBACK

PURPOSE Test a GSI I/O LAM (IOL) module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

/LOOPBACK	Signals, that the external loopback connections have been established and may be used in the test.
FUNCTION	This test performs some manipulations with a IOL module which allow a test with a scope. The full test algorithm works as follows:
EXAMPLE	TEST GSI_IOL C=1 N=12 /START
REMARKS	The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.
Action rout.	I$CTGSI_IOL
Author	Walter F.J. Mueller

Remarks

File name	I$CTGSI_IOL.PPL
Dataset	-
REMARKS	-

Description

CALLING

```
@CALL I$CTGSI_IOL(I_B,I_C,I_N,F_DELTA,C_OPTIONS,
                  C_LOOPBACK);
```

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

C_LOOPBACK CHAR(*) VAR [INPUT]
/LOOPBACK qualifier. Signals, that the external loopback connections have been established and may be used in the test.

FUNCTION This procedures performs a selftest for a GSI IOL module. For a more detailed discussion look in the description of the command TEST GSI_IOL.

REMARKS -

EXAMPLE @CALL I$CTGSI_IOL(0,1,12,0.E0,'/RUN',");

TEST LRS_2228

TEST LRS_2228
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP
/VALUE

PURPOSE Test a LRS 2228 TDC module.

PARAMETERS

B=b	Number of the branch of the module to be tested. Replaceable default = 0
C=c	Number of the crate of the module to be tested. Replaceable default = 1
N=n	Station number of the module to be tested. Replaceable default = 1
REPEAT=r	Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.
/LIST	Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.
/STOP	Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.
/RUN	Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.
/START	Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.
/LOOP	Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

/VALUE The values of the test conversion are written to SYS$OUTPUT.

FUNCTION This test exploits the F(25) self test function of a LRS 2228 TDC to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the module with a F(9) function, read all channels and check that they contain a zero.

Perform a test conversion with a F(25) function, wait for the conversion by doing 60 CAMAC accesses, read all channels with a F(2) function and check that they contains neither a zero nor an overflow value. If the /VALUE qualifier has been specified, the test conversion values are written to SYS$OUTPUT and may be used to check the homogenety of the conversion factor.

Read again all channels and check that the previous F(2)A(7) read has actually zeroed all channels.

EXAMPLE TEST LRS_2228 C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTLRS_2228

Author Walter F.J. Mueller

Remarks

File name I$CTLRS_2228.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$CTLRS_2228(I_B,I_C,I_N,F_DELTA,C_OPTIONS, C_VALUE);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

C_VALUE CHAR(*) VAR [INPUT]
/VALUE qualifier. If specified, the values of the test conversion are written to SYS$OUTPUT.

FUNCTION This procedures performs a selftest for a LRS 2228 TDC. For a more detailed discussion look in the description of the command TEST LRS_2228.

REMARKS -

EXAMPLE @CALL I$CTLRS_2228(0,1,12,0.E0,'/RUN','/VALUE');

TEST LRS_2249

TEST LRS_2249
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP
/PEDESTAL

PURPOSE Test a LRS 2249 ADC module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

/PEDESTAL The values of the pedestal and calibration values of the test conversion are written to SYS$OUTPUT.

FUNCTION This test exploits the F(25) self test function of a LRS 2249 ADC to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the module with a F(9) function, read all channels and check that they contain a zero.

Perform a test conversion with a F(25) function. This will trigger a conversion with an internally generated gate but no input current thus yielding a pedestal value. Wait for the conversion by doing 60 CAMAC accesses, read all channels with a F(2) function and check that they contain a reasonable pedestal value.

Set the dataway INHIBIT and perform another test conversion with a F(25) function. This will trigger a conversion with an internally generated test current thus yielding a rough estimate of the channel calibration. Again wait for the conversion by doing 60 CAMAC accesses, read all channels with a F(2) function and check that they contain a reasonable value.
If the /PEDESTAL qualifier has been specified, the the pedestal and test conversion values are written to SYS$OUTPUT. Be aware, that the pedestal values are for the internally generated gate with and may different in the experimental setup!

Read again all channels and check that the previous F(2)A(11) read has actually zeroed all channels.

EXAMPLE TEST LRS_2249 C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTLRS_2249

Author Walter F.J. Mueller

Remarks

File name I$CTLRS_2249.PPL

Dataset -

Description

CALLING @CALL I$CTLRS_2249(I_B,I_C,I_N,F_DELTA,C_OPTIONS, C_PEDESTAL);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

C_PEDESTAL CHAR(*) VAR [INPUT]
/PEDESTAL qualifier. If specified, the pedestal and calibration values of the test conversion are written to SYS$OUTPUT.

FUNCTION This procedures performs a selftest for a LRS 2249 ADC. For a more detailed discussion look in the description of the command TEST LRS_2249.

REMARKS -

EXAMPLE @CALL I$CTLRS_2249(0,1,12,0.E0,'/RUN','PEDESTAL');

TEST LRS_2551

TEST LRS_2551
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP

PURPOSE Test a LRS 2551 scaler module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

FUNCTION This test exploits the F(25) self test function of a LRS 2551 scaler to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the module with a F(9) function, read all channels and check that they contain a zero.

Issue a test with a F(25) function, read all channels and check whether they have been incremented once.

Issue further 32 tests with a F(25) function, read all channels and check whether they contain now the value 33.

Clear the module with a F(9) function, read all channels and check that they contain again zero.

EXAMPLE TEST LRS_2551 C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTLRS_2551

Author Walter F.J. Mueller

Remarks

File name I$CTLRS_2551.PPL

Dataset -

Description

CALLING @CALL I$CTLRS_2551(I_B,I_C,I_N,F_DELTA,C_OPTIONS);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

FUNCTION This procedures performs a selftest for a LRS 2551 scaler. For a more detailed discussion look in the description of the command TEST LRS_2551.

REMARKS -

EXAMPLE @CALL I$CTLRS_2551(0,1,12,0.E0,'/RUN');

TEST LRS_4432

TEST LRS_4432
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP

PURPOSE Test a LRS 4432 scaler module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

FUNCTION This test exploits the F(25) self test function of a LRS 4432 scaler to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the module by writing the clear bit in the CSR, than load and read all channels by writing the load and read bit in the CSR and reading 32 words. Check, that all channels are zeroed.

Issue a test by writing the test bit in the CSR, load and read all channels and check whether all bytes of all channels have been incremented once. For details of the test function look in the 4432 manual.

Issue further 254 tests, load and read all channels. Check, the all channels return a word with all 24 bits set. (Each byte has been incremented 255 times and thus should contain '11111111'B).

Clear the module, load and read all channels. Check, that all channels are again zeroed.

EXAMPLE TEST LRS_4432 C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTLRS_4432

Author Walter F.J. Mueller

Remarks

File name I$CTLRS_4432.PPL

Dataset -

Description

CALLING @CALL I$CTLRS_4432(I_B,I_C,I_N,F_DELTA,C_OPTIONS);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

FUNCTION This procedures performs a selftest for a LRS 4432 scaler. For a more detailed discussion look in the description of the command TEST LRS_4432.

REMARKS -

EXAMPLE @CALL I$CTLRS_4432(0,1,12,0.E0,'/RUN');

TEST LRS_4434

TEST LRS_4434
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP

PURPOSE Test a LRS 4434 scaler module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

FUNCTION This test exploits the F(25) self test function of a LRS 4434 scaler to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the module by writing the clear bit in the CSR, than load and read all channels by writing the load and read bit in the CSR and reading 32 words. Check, that all channels are zeroed.

Issue a test by writing the test bit in the CSR, load and read all channels and check whether all bytes of all channels have been incremented once. For details of the test function look in the 4432 manual.

Issue further 254 tests, load and read all channels. Check, the all channels return a word with all 24 bits set. (Each byte has been incremented 255 times and thus should contain '11111111'B).

Clear the module, load and read all channels. Check, that all channels are again zeroed.

Action rout. I$CTLRS_4434

Author Walter F.J. Mueller

Remarks

File name I$CTLRS_4432.PPL

Dataset -

REMARKS -

EXAMPLE TEST LRS_4434 C=1 N=12 /START

Description

CALLING @CALL I$CTLRS_4434(I_B,I_C,I_N,F_DELTA,C_OPTIONS);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C	BIN FIXED(15) [INPUT] Crate number of the module to be tested.
I_N	BIN FIXED(15) [INPUT] Station number of the module to be tested.
F_DELTA	BIN FLOAT(24) [INPUT] Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..
C_OPTION	CHAR(*) VAR [INPUT] Test option qualifier set, expected values are: /LIST/STOP /RUN/START/LOOP
FUNCTION	This procedures performs a selftest for a LRS 4434 scaler. For a more detailed discussion look in the description of the command TEST LRS_4434.
REMARKS	-
EXAMPLE	@CALL I$CTLRS_4434(0,1,12,0.E0,'/RUN');

TEST MPI_BIT

TEST MPI_BIT
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP

PURPOSE Test a MPI bit encoder module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

FUNCTION This command performs a simple test for a MPI bit encoder module. The full test algorithm works as follows:

Generate a test pattern and write it to the bit encoder.

Read back the bit multiplicity and 18 bit numbers. Than check, whether the multiplicity and the bit numbers are correct and whether the correct stop word has been generated by the bit encoder.

EXAMPLE TEST MPI_BIT C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTMPI_BIT

Author Walter F.J. Mueller

Remarks

File name I$CTMPI_BIT.PPL

Dataset -

Description

CALLING @CALL I$CTMPI_BIT(I_B,I_C,I_N,F_DELTA,C_OPTIONS);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

FUNCTION This procedures performs a selftest for a MPI bit encoder. For a more detailed discussion look in the description of the command TEST MPI_BIT.

REMARKS -

EXAMPLE @CALL I$CTMPI_BIT(0,1,12,0.E0,'/RUN');

TEST MPI_TDC

TEST MPI_TDC
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP
/VALUE

PURPOSE Test a MPI slow TDC module.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

/VALUE The values of the time intervall between clear and strobe is written to SYS$OUTPUT.

FUNCTION This test exploits the F(16)A(0,1) self test functions of a MPI slow TDC to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the TIME with a F(16)A(1) function.

Strobe the TIME with a F(16)A(0) function.

Read the TIME with a F(0)A(0) function and check whether a reasonable value has been returned. If the /VALUE qualifier has been specified, this time is written to SYS$OUTPUT. Because this is the time between the clear and the strobe action done with a single I$CFGA call, is is independant of the speed and load of the host VAX but a measure of the MBD performance.

EXAMPLE TEST MPI_TDC C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTMPI_TDC

Author Walter F.J. Mueller

Remarks

File name I$CTMPI_TDC.PPL

Dataset -

Description

CALLING @CALL I$CTMPI_TDC(I_B,I_C,I_N,F_DELTA,C_OPTIONS, C_VALUE);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

C_VALUE CHAR(*) VAR [INPUT]
/VALUE qualifier. If specified, the time intervall between clear and strobe is written to SYS$OUTPUT.

FUNCTION This procedures performs a selftest for a MPI slow TDC. For a more detailed discussion look in the description of the command TEST MPI_TDC.

REMARKS -

EXAMPLE @CALL I$CTMPI_TDC(0,1,12,0.E0,'/RUN','/VALUE');

TEST REGISTER

TEST REGISTER
B=b C=c N=n
A=a F=f
REPEAT=r
/LIST /STOP /RUN /START /LOOP
WIDTH=w /FULL

PURPOSE Perform tests with a any register in a CAMAC module.

PARAMETERS

B=b Number of the branch of the module to be tested.

C=c Number of the crate of the module to be tested.

N=n Station number of the module to be tested.

A=a Number of the subaddress in the module to be tested.

F=f Function in the module to be tested.

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed.

WIDTH=w The width of the register to be tested in bits. May be between 2 and 24, the default is 16.

/FULL A full set of test patterns is written and read back in every test cycle. The default is one test pattern per cycle. The /FULL mode is usefull to check for infrequent, intermittent problems.

FUNCTION This test uses any read/write register in a CAMAC module as a loopback mirror and tries to verify the datatransfer thru the branch driver, the branch branch cables, the crate controller, the dataway, the module register and back. This is done by sending test patterns and reading them back. The full test algorithm works as follows:

Generate a new test pattern (for details of look in the description of I$GTPAT).

Send the test pattern to the module, use the specified subaddress and the function code plus 16. Read the pattern back immediately with the same subaddress and the specified function code. Compare send and read pattern and check whether read and write had X and Q response. If the /FULL qualifier has been specified, a full set of test patterns is generated and processed in all test cycles.

Action rout. I$CTREGISTER

Author Walter F.J. Mueller

Remarks

File name I$CTREGISTER.PPL

Dataset -

REMARKS -

EXAMPLE TEST REGISTER C=1 N=22 /START

Description

CALLING

```
@CALL I$CTREGISTER(I_B,I_C,I_N,I_A,I_F,
        F_DELTA,C_OPTION,I_WIDTH,C_FULL);
```

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

I_A BIN FIXED(15) [INPUT]
Subaddress of the module to be tested.

I_F BIN FIXED(15) [INPUT]
Function code in the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

I_WIDTH BIN FIXED(15) [INPUT]
Width of the register to be tested in bits. May be between 2 and 24.

C_FULL CHAR(*) VAR [INPUT]
/FULL qualifier. If specified, all test patters are written and read in every test interation.

FUNCTION This procedure performs dataway loopback tests with a any read/write register in a CAMAC module. For a more detailed discussion look in the description of the command TEST REGISTER.

REMARKS -

EXAMPLE

```
@CALL I$CTREGISTER(0,1,22,0,2,0.E0,
        '/RUN',16,'/FULL');
```

TEST SEN_2047

TEST SEN_2047
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP

PURPOSE Test a SEN 2047 pattern unit.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

FUNCTION This test exploits the F(25) self test function of a SEN 2047 pattern to perform a simple go/nogo test. The full test algorithm works as follows:

Clear the module with a F(9) function, read the pattern and check that it contains a zero.

Issue a test with a F(25) function, read the pattern with a F(0) function and check whether it contains all one's.

Reread the pattern with a F(2) and a F(0) function and check whether is still contains an all one's pattern or is zeroed respectively.

Finally issue another F(0) on even test cycles and a F(25) on odd test cycles. This causes the pattern units LED's to blink and may be used to signal test activities for this crate.

EXAMPLE TEST SEN_2047 C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTSEN_2047

Author Walter F.J. Mueller

Remarks

File name I$CTSEN_2047.PPL

Dataset -

Description

CALLING @CALL I$CTSEN_2047(I_B,I_C,I_N,F_DELTA,C_OPTIONS);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

FUNCTION This procedures performs a self-test for a SEN 2047 pattern unit. For a more detailed discussion look in the description of the command TEST SEN_2047.

REMARKS -

EXAMPLE @CALL I$CTSEN_2047(0,1,12,0.E0,'/RUN');

TEST SEN_2090

TEST SEN_2090
B=b C=c N=n
REPEAT=r
/LIST /STOP /RUN /START /LOOP

PURPOSE Test a SEN 2090 video display driver.

PARAMETERS

B=b Number of the branch of the module to be tested. Replaceable default = 0

C=c Number of the crate of the module to be tested. Replaceable default = 1

N=n Station number of the module to be tested. Replaceable default = 1

REPEAT=r Repetition time in seconds for started tests. Must be between 0. and 3599., the default value is 1 sec.

/LIST Will list tests of the given type. List all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/STOP Will abort test of the given type. Abort all tests if no crate or station has been specified or just the tests in a given crate and/or station.

/RUN Will execute the test on the specified CAMAC address only one time. This is the default if on other qualifier has been specified.

/START Will start the repetive execution of the test on the specified CAMAC address at time intervalls specified with the REPEAT parameter.

/LOOP Like /START, but the test is executed as often as possible, limitted only by the CPU and CAMAC speed. This LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

FUNCTION This test writes patterns to a SEN 2090 video display driver which allows a visual go/nogo test. The full test algorithm works as follows:

In the first test cycle the module is initialised by enabling the character mode and the cursor and by clearing the screen.

Than a pattern consisting of all printable characters is written to the screen. The pattern is shifted by one character in each line and in each test cycle.

EXAMPLE TEST SEN_2090 C=1 N=12 /START

REMARKS The LOOP function will only work if the GOOSY command menu is not active. Leave the command menu after submitting of the command.

Action rout. I$CTSEN_2090

Author Walter F.J. Mueller

Remarks

File name I$CTSEN_2090.PPL

Dataset -

REMARKS -

Description

CALLING @CALL I$CTSEN_2090(I_B,I_C,I_N,F_DELTA,C_OPTIONS);

ARGUMENTS

I_B BIN FIXED(15) [INPUT]
Branch number of the module to be tested.

I_C BIN FIXED(15) [INPUT]
Crate number of the module to be tested.

I_N BIN FIXED(15) [INPUT]
Station number of the module to be tested.

F_DELTA BIN FLOAT(24) [INPUT]
Repetition time in seconds for started tests. Must be between 0. and 3599., a recommended value is 1..

C_OPTION CHAR(*) VAR [INPUT]
Test option qualifier set, expected values are:
/LIST/STOP /RUN/START/LOOP

FUNCTION This procedures performs a self test for a SEN 2090 video display driver. For a more detailed discussion look in the description of the command TEST SEN_2090.

REMARKS -

EXAMPLE @CALL I$CTSEN_2090(0,1,12,0.E0,'/RUN');

TYPE BUFFER

TYPE BUFFER number /HEADER

PURPOSE	Start to type data buffers
PARAMETERS	
number	integer default=1 Number of buffers to be typed .
/HEADER	Output header only.
EXAMPLE	TYPE BUF 1
NOTE	This command works only if buffer checking is enabled.

Description

FUNCTION	Sets the number of data buffers to be dumped by I$ACQ_DUMP.
File name	I$ACQ_DMP_BUF.PPL
Action rout.	I$ACQ_DMP_BUF
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

TYPE EVENT

TYPE EVENT number id /SAMPLE /HEADER

PURPOSE	Start to type events
PARAMETERS	
number	integer default=1 Number of events to be typed.
id	integer default=0 Optional subevent id
/SAMPLE	Type one (first) event per buffer.
/HEADER	Only headers are output
EXAMPLE	TYPE EVE 10
NOTE	This command works only if buffer checking is enabled.

Description

FUNCTION	Start event typing.
File name	I$ACQ_DMP_EVT.PPL
Action rout.	I$ACQ_DMP_EVT
Dataset	-
Version	1.01
Author	Walter F.J. Mueller
Last Update	12-APR-1985

TYPE FILE

TYPE FILE file skip buffers events id outfile
/HEADER /DATA
/EVENTHEADER
/SAMPLE
/PRINT

PURPOSE Output GOOSY list mode data file (called in MUTIL).

PARAMETERS

file required string replace
List mode data file.

skip integer default=0
Optional number of buffers to skip.

buffers integer default=1
Optional number of buffers to output

events integer default=10
Optional number of events to output

id integer default= 0
Optional number of events to output

output string replace
Optional output file.

/HEADER switch
Output GOOSY file header only.

/DATA switch
Output formatted data.

/SAMPLE switch
Output one event per buffer. Valid for /DATA.

/EVENTHEADER switch
Output event header only. Valid for /DATA.

/PRINT switch
Print output file. Valid for /DATA.

Caller MUTIL

Author H.G.Essel

Example

```
$ MUTIL TYPE FIL X.LMD OUT=X.HEAD /HEAD
file header is written into X.HEAD.
$ MUTIL TYPE FIL X.LMD 10 1 Y.LIS /DATA
Write 11th buffer of X.LMD to Y.LIS in ASCII.
```

Remarks

File name I$FILCM.PPL

Created by I$FILCM.PPL

Description

CALLING
```
STS=I$FIL_T(CV_file,L_skip,L_buffers,L_events,L_id,
            CV_outfile,CV_output,
            I_sample,I_header,I_print)
```

COMMAND
```
TYPE FILE file skip buffers events id outfile
            /HEADER /DATA
            /EVENTHEADER
            /SAMPLE
            /PRINT
   Arguments/Parameters description
```

FILE

Routine arg. Input CHAR(*) VAR

Command par. required string replace
File name for input.

SKIP

Routine arg. Input BIN FIXED(31)

Command par. integer default=0
Optional number of buffers to be skipped.

BUFFERS

Routine arg. Input BIN FIXED(31)

Command par. integer default=1
Optional number of buffers to be output.

EVENTS

Routine arg. Input BIN FIXED(31)

Command par. integer default=10
Optional number of events to be output.

ID

Routine arg. Input BIN FIXED(31)

Command par. integer default=0
Optional number of subevent to be output (FEP id).

OUTFILE

Routine arg. Input CHAR(*) VAR

Command par. string replace
Optional file name for output.

/OUTPUT

Routine arg. Input CHAR(*) VAR

Command par. switch default=/DATA

/HEADER	Output header only.
/DATA	Output formatted data

/SAMPLE

Routine arg. Input BIN FIXED(15) valid values 0,1

Command par. switch

Output one event per buffer

/EVENTHEADER

Routine arg. Input BIN FIXED(15) valid values 0,1

Command par. switch

Output event header only

/PRINT

Routine arg. Input BIN FIXED(15) valid values 0,1

Command par. switch

Print output file

Function

Read specified input file and output GOOSY file header and data. Only standard GOOSY data formats are supported.

Appendix B

DCL Commands

ALIAS

ALIAS command arguments

PURPOSE Handle GOOSY alias command names

ARGUMENTS

command subcommand key:
CREATE
DELETE
SHOW
A quotation mark enters menu.

arguments argument list depending on command.

Description

FUNCTION Alias names can be defined on two levels:
Environment and Global.
Alias names defined for a certain environment
are activated together with the CRENV command and deactivated with the DLENV command. Global alias names are active anytime. An alias is searched first in the environment table and then in the global table. Alias names are deleted during logout.

Version 1.01

Author H.G.Essel

Last Update 14-APR-1987

CREATE

CALLING ALIAS CREATE name string environment /GLOBAL
CRALIAS name string environment /GLOBAL

name Name of the alias. May contain alphanumeric characters including _ $.
A quotation mark enters menu.

string	Replacement string for alias. If the string contains delimiters (in terms of DCL arguments) it must be enclosed in quotes.
environment	Optional environment for which the alias will be created. If omitted a global alias is created.
/GLOBAL	Create global alias. Ignore environment parameter.

DELETE

CALLING	ALIAS DELETE name environment /GLOBAL DLALIAS name environment /GLOBAL
name	Name of the alias. May contain alphanumeric characters including _ $. A quotation mark enters menu.
environment	Optional environment for which the alias will be deleted. If omitted a global alias is deleted.
/GLOBAL	Delete global alias. Ignore environment parameter.

SHOW

CALLING	ALIAS SHOW name environment /GLOBAL/ACTIVE SHALIAS name environment /GLOBAL/ACTIVE
name	Optional name of the alias. A quotation mark enters menu.
environment	Optional environment for which the alias will be listed. If not specified, /ACTIVE is assumed.
/GLOBAL	Only global alias names are listed.
/ACTIVE	All active alias names are listed.

ATENVIR

ATENV*IR environment

PURPOSE	Attach environment.
FUNCTION	Defines logical GOOSY_PROMPT to GOO_env_MBX. Sets prompter GOOSY to MGOOTP1. The environment must be started by CRENVIR /MBX at the same node.
Version	1.01
Author	H.G.essel
Last Update	14-APR-1987

CHANAL

CHANAL infile outfile /COPY/FULL

PURPOSE	Check GOOSY analysis programs.
ARGUMENTS	
infile	Input file with analysis routine.
outfile	Output file for new analysis routine (/COPY).
/COPY	Copy input to output file. The type specifications are inserted in the $LOC macro calls if not there.
/FULL	List lines which are modified

Description

FUNCTION	Calls MANALCH to check calling $ACCU and $COND macros. Condition types and spectrum data types are checked in the $LOC macros. The input file is copied to the output file if /COPY is given. Otherwise only a check is done.
Version	1.01
Author	H.G.Essel
Last Update	10-JAN-1989

CLINK

CL file/switches /switches

PURPOSE Link programs. Defaults are updated.

ARGUMENTS

file Name of object file (def.=last)

switches Linker switches. Behind SPACE are not updated

DESCRIPTION

FUNCTION Two problems are solved:
1. To keep last input as default
2. To inculde standard options.

There are two ways to specify switches. Directly following the file name the switches are kept as default for the next call. After a blank switches are temporary used. If one wants to use the last file, but with additional switches, the file name must be *.

REMARKS -

File name CLINK.COM

Dataset -

Version 1.01

Author H.Essel

Last Update 20-JAN-1984

COMMENT

COMMENT topic command arguments

PURPOSE Write and read comments for a topic.

ARGUMENTS

topic Name of the COMMENT topic. A logical name must be defined for the topic file:
CMT$topic = filespec

command subcommand key:
CREATE
READ [default]
WRITE
EDIT
KEYS

arguments argument list depending on command.

Description

FUNCTION COMMENT writes and reads a comment file. The file must be created once. A logical name must be assigned in the form
CMT$topic = filespec
Entries are written with date and username.
Entries are displayed in a specified date window.

Version 1.01

Author H.G.Essel

Last Update 10-MAR-1991

CREATE

CALLING COMMENT topic CREATE

topic Name of the COMMENT topic. A logical name must be defined for the topic file. Headlines are prompted line by line. These lines are displayed with the READ command.

READ

CALLING COMMENT topic [READ]keylist begin-date end-date

topic Name of the COMMENT topic. A logical name must be defined for the topic file.

READ If no argument follows, this keyword may be omitted.

keylist List of keys. Only entries marked with specified keys are output. * means all keys and defaults begin-date to 1-MAR-1991. Not specified defaults begin-date to YESTERDAY.

date Date from/to wich file entries are to be displayed. Defaults depend on keylist. Begin-date default is YESTERDAY when keylist is empty, and 1-MAR-1991 when keylist is not empty (but may be *). End-date defaults to TOMORROW, if not specified.

WRITE-EDIT

CALLING COMMENT topic WRITE-or-EDIT keylist

topic Name of the COMMENT topic. A logical name must be defined for the topic file.

WRITE Lines are prompted and are inserted in the file.

EDIT The headline is written to a temporary file and the editor in invoked.

keylist List of keys. Keys mey be specified as filter when reading entries.

KEYS

CALLING COMMENT topic KEYS

topic Name of the COMMENT topic. A logical name must be defined for the topic file. List all topics in a topic file.

Format

The file is a normal text file and may be edited. The syntax is:

Lines starting with ! are displayed also outside
time window. Example: header lines.
Lines starting with - are date lines.
All others are text lines.

Example

Creating a new topic file for topic MYCOM:

```
$ DEFINE CMT$MYCOM SYS$LOGIN:CMT_MYCOM.TXT
$ COMM MYCOM CREATE
> ...
$ ...
```

Insert entry:

```
$ COMM MYCOM W
> ...
$ ...
```

Insert entry with key (use editor):

```
$ COMM MYCOM EDIT key1,key2
> ...
$ ...
```

Read entries with key (no date limit):

```
$ COMM MYCOM READ key1
$ ...
```

Read all entries:

```
$ COMM MYCOM READ *
$ ...
```

Read entries with key since yesterday:

```
$ COMM MYCOM READ key1 yesterday
$ ...
```

Read all entries since yesterday:

```
$ COMM MYCOM READ
$ ...
```

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

PURPOSE	General compile procedure for all compilers THE PREVIOUS VERSION CAN BE CALLED BY OCOMPILE !
ARGUMENTS	
file	Default extension from symbol defcompi. Wildcarding supported.
/PREQUAL	List of precompiler qualifiers
/QUALIFIER	List of compiler qualifiers
list	List of items separated by commas.
/GIPSY	List of GIPSY qualifiers
list	List of items separated by commas.
/LIBRARY	List of private Libraries
list	List of items separated by commas.
/COMPILE	Compile in any case
/DEBUG	Compile with debug switch set
/KEEP	Hold temporary source
/OLB=	Private object library
/SINCE=	Compile only those sources dated later than "time"
/BEFORE=	Compile only those sources dated ealrier than "time"
time	Specify an absolute time or a combination of absolute and delta time.

/NEWPLI	Use the new PL/I Compiler
/CALL	Compile option for GOOSY Macros: Expand macros to subroutine calls
/MACRO	Compile option for GOOSY Macros: Expand macros to PL/1 code
/FAST	Compile option for GOOSY Macros: Expand macros to fast PL/1 code
/BATCH	Compile in batch job in queue SYS$COMP.
Example	COM ABC /QUAL=(LIST,SHOW=INCLUDE) COM ABC /QUAL=(LIST,SHOW=(INCLUDE,MAP))
Languages	For the following file types the appropriate compilers are called: PPL, PLI, MOD, PAS , DEF, MSG, DAR, FOR, MAR, MAC, C, MBD, COM, TEX, FONT, PGIP, CGIP, DGIP, CPGIP

DESCRIPTION

CALLING

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/DIR=d

ARGUMENTS

file	File name, wildcards included, specifies one or more files to be compiled. The default extension is taken from default compiler (your global symbol "defcompi"). With an "*" you get your last input.
/PREQUAL	List of precompiler qualifiers This list will be passed to the precompiler,if used.
/QUALIFIER	List of compiler qualifiers This list will be passed to the called compiler.
list	List of items separated by commas. An item may be a sublist like /QUAL=(list).
/GIPSY	List of GIPSY qualifiers This list will be passed to GIPSY.
list	List of items separated by commas. An item may be a sublist like /GIPSY=(list).

/LIBRARY	List of private Libraries Each item of this list will be added to the file name with the qualifier "/LIB".
list	List of up to three items separated by commas.
/COMPILE	Compile in any case
/DEBUG	Compile with debug switch set, added to /QUAL PLI: /DEB=ALL/CHECK/SHOW=(INCL,MAP,EXPANS, SOURCE,TERMINAL,TRACE)/NOOPTIMIZE FOR: /DEB/NOOPT MAR: /DEB
/KEEP	hold temporary source (output of precompiler) FORTRAN precompiler generates .FORTEMP, PLI precompiler .PLITEMP files.
/OLB=	Private object library
/DIRECTORY=	Set default directory.
/SINCE=	Compile only those sources dated later than "time"
/BEFORE=	Compile only those sources dated ealrier than "time"
time	Specify an absolute time or a combination of absolute and delta time. See section 2.5 in the VAX/VMS DCL dictionary for details on specifying times or access the HELP topic SPECIFY.
/NEWPLI	Use the new PL/I Compiler
/CALL	Compile option for GOOSY Macros: Expand macros to subroutine calls
/MACRO	Compile option for GOOSY Macros: Expand macros to PL/1 code
/FAST	Compile option for GOOSY Macros: Expand macros to fast PL/1 code (No checks of bits, no counter increments).
/BATCH	Compile in batch job in queue SYS$COMP.
FUNCTION	There are three features in this procedure: 1. Compilation only if the source is younger than the object

Depending on the specified file extension, different compilers are called. If /COM is specified, the file is compiled in any case. The prequalifier, enclosed in brackets, are passed to a precompiler, if used. Compiler qualifiers are not specified as normal!!: all qualifiers have no slashes and are seperated by commas e.g. /QUAL=(NOOPT,LIST,NOOBJECT). 2. Use wild cards in file name to compile more than one file. Use /SINCE and/or /BEFORE switch to compile only sources of a specified time intervall. 3. To keep the last input as default If the first parameter (file name) is only an *, the last specified input is used. All switches directly following the the file name are kept as default for the next call. After a blank switches are temporary used.

Examples

COM MGENHEAD.PPL/PRE=(V) /KEEP/COM calls precomposer MPRECOMP with tagword V and then PLI compiler. /KEEP option holds MGENHEAD.PLITEMP. Compilation is done without respect to the dates of source and object file. COM */DEB /COM Compiles again MGENHEAD.PPL/PRE=(V) /DEB/COM COM ABC /QUAL=(LIST,SHOW=(INCLUDE)) RSX-MACRO compilation with macro expension listing COM XYZ.MAC /QUAL=(LIST,MACRO)

Compiler

FORTRAN

input File type .FOR compiled by FORTRAN

input File type .DAR calls preprocessor and generates FORTRAN file .FORTEMP. Then FORTRAN is called.

PLI

input File type .PLI compiled by CPLI

input File type .PPL calls preprocessor and generates PLI file .PLITEMP. Then CPLI is called.

C

input File type .C compiled by CC

GIPSY

input File type .DGIP compiled by GIPSY to file .COM

input File type .PGIP calls preprocessor MPRECOM and generates GIPSY file .PGIPTEMP. Then GIPSY generates .PLITEMP. Then CPLI is called.

input File type .CGIP compiled by GIPSY to file .CGIPTEMP. Then CC is called.

input File type .CPGIP compiled by GIPSY to file .CPGIPTEMP. Then CP020 is called.

MODULA

input File type .MOD compiled by MODULA

MESSAGE

input File type .MSG is processed by preprocessor MPREMES which generates file .MSGTEMP. Then MESSAGE is called.

MACRO

input File type .MAC compiled by CRSXMAC for RSX

input File type .MAR compiled by VAX MACRO

PASCAL

input File type .PAS compiled by PASCAL

MBD

input File type .MBD compiles by CMBDCOM

LATEX

input File type .TEX compiled by CTEXCOM

CONCAT

CONCAT infile outfile /LOG/DELETE/CONFIRM/APPEND

PURPOSE	Concatenates input files to one output file.
ARGUMENTS	
infile	Input file spec. The version number must be *. All files are concatenated in the correct order (Version n+1 behind version n).
outfile	Output file name. Must be different from input.
/APPEND	If output file already exists, input files are appended.
/LOG	Log operations
/DELETE	Delete copied files
/CONFIRM	Confirm file deletion (not copy).

Description

FUNCTION	When several versions of a file are copied into one file, the highest version is on top. If this is not wanted, CONCAT copies the files in reverse order: Lowest version is on top. Lowest version is assumed to be 1. Highest version is highest on disk. Versions may be missing.
EXAMPLE	X.DAT;5 X.DAT;4 X.DAT;2 CONCAT X.DAT;* Y.DAT copies X.DAT;2 to Y.DAT and appends then X.DAT;4 and X.DAT;5
Version	1.01
Author	H.G.Essel
Last Update	21-OCT-1988

CREDB

CREDB basename filename size[KB]
/DYNLISTS=d /SPECTRA=s /CONDITIONS=c
/PICTURES=p /DIRECTORIES=d /POOLS=p
/POLYGONS=p /NEW /SAVE=file

PURPOSE Create an preformat a new GOOSY data base.

ARGUMENTS

basename Name of data base.

filename File specification for data base file. The name must be equal the data base name, the file type should be .SEC.

size Size of data base in Kilobytes. (1 VMS page is .5KB)

/DYNLISTS=d Maximum number of dynamic lists. (def=10)

/SPECTRA=s Maximum number of spectra (def=100)

/CONDITIONS=c Maximum number of conditions (def=100)

/PICTURE=p Maximum number of pictures and frames (def=100)
Each frame takes one entry!

/DIRECTORIES=d Maximum number of directories (def=20)

/POOLS=p Maximum number of pools (def=20)

/POLYGONS=p Maximum number of polygons (def=20)

/NEW An old existing data base is deleted and a new one is created. If not specified, an error is given.

/SAVE=file Save command procedure to create data base in file.

Description

CALLING CREDB basename filename size[KB]
/DYNLISTS=d /SPECTRA=s /CONDITIONS=c
/PICTURES=p /DIRECTORIES=d /POOLS=p
/POLYGONS=p /NEW /SAVE=file

ARGUMENTS

basename Name of data base.

filename File specification for data base file. The name must be equal the data base name, the file type should be .SEC.

size Size of data base in Kilobytes. (1 VMS page is .5KB)

/DYNLISTS=d Maximum number of dynamic lists. (def=10)

/SPECTRA=s Maximum number of spectra (def=100). The bit table for spectra is created with this number of entries.

/CONDITIONS=c Maximum number of conditions (def=100) The bit table for conditions is created with this number of entries.

/PICTURE=p Maximum number of pictures and frames (def=100)
Each frame takes one entry!

/DIRECTORIES=d Maximum number of directories (def=20)

/POOLS=p Maximum number of pools (def=20)

/POLYGONS=p Maximum number of polygons (def=20)

/NEW An old existing data base is deleted and a new one is created. If not specified, an error is given.

/SAVE=file Save command procedure to create data base in file. If the base already exists, the command procedure is written, but not executed.

FUNCTION A new GOOSY data base is created. The default directories for spectra, conditions, pictures and dynamic lists are created.
A directory and pool for user data elements is
created, both named DATA. The directory has 100 slots.

REMARKS The data base file must not exist (except /NEW or /SAVE=file is specified).

EXAMPLE

```
CREDB DB DB.SEC 5000 /SPEC=200
  CREDB DB DB.SEC 5000 /SPEC=200/SAVE=DB
    save command procedure in DB.COM. If DB.SEC exists,
    the command procedure is not executed.
```

CRENVIR

CRENV*IR environment program component
/ONLINE/OFFLINE/DEFAULT
/$TMR/$ANL/$DSP/$DBM/J11
/[NO]DECWINDOW
/PRIORITY=p/DELETE

PURPOSE Creates a GOOSY environment and optional some GOOSY standard components

ARGUMENTS

environment Name of the environment (max. 4 char)

program Optional name of private analysis program. If not specified, MGOOANL is assumed. This private program is started by /$ANL or by /ONLINE or /OFFLINE, if no /DEF is specified.

component Optional component name for analysis program. Default is $ANL.

/ONLINE Creates TMR, DSP, DBM and analysis program specified by program (default=MGOOANL, if /DEF is not specified). If /DEF is specified, a GOOSY standard analysis program is started. If program is specified, this is used regardless of /DEFAULT.

/OFFLINE Creates DSP, DBM and analysis program specified by program (default=MGOOANL, if /DEF is not specified). If /DEF is specified, a GOOSY standard analysis program is started. If program is specified, this is used regardless of /DEFAULT.

/DEFAULT Creates default analysis. Otherwise use specified program, which is the name of your private analysis program (default=MGOOANL). If a program name has been specified, this switch is ignored.

/$TMR Create Transport Manager $TMR

/$ANL Create Analysis program $ANL

/$DSP	Create Display $DSP
/NODECWINDOW	Use old version of display (/$DSP must be given) Default is the DEC-GKS version.
/$DBM	Create Data Base Manager $DBM
/J11	Create standard analysis program GOO$EXE:MGOOANL
/PRIORITY=	Specify priority for analysis component (DEF=3). You cannot raise the priority above 4 if you have not the proper privileges.
/DELETE	Delete environment log files.

Description

FUNCTION

A GOOSY environment is created only if it does not already exist. The components specified by qualifiers are created. You may use this command to create components in an existing environment. Specifiy the current name or * for the environment parameter in this case.

A user specific analysis program linked by command LANL is started from current directory by /$ANL.

GOOSY provides a standard analysis program. This analysis is started by /J11 or by /DEFAULT.

By default the analysis programs are started with priority 3 unless /PRIO=p is specified.

NOTE

An environment is deleted by command DLENV.

Version 1.01

Author H.G.Essel

Created 14-JAN-1987

Last Update 19-OCT-1987
File creation error handled /HE
18-APR-1990
create enviroment table in this module /RF
23-Jul-1993
DEC-GKS V5.0 version is default. /HE

Examples

```
$  CRENV SUSI /$DBM ! create environment and $DBM
$  CRENV SUSI /$TMR/$ANL ! add $ANL component
$  DLENV ! delete environment
$  CRENV SUSI /ONLINE/DEF ! Create $TMR, $DBM, $DSP, $ANL
                          ! Use GOO$EXE:MGOOANL
$  DLENV ! delete environment
$  CRENV SUSI /OFFLINE ! Create $DBM, $DSP, $ANL
                       ! Use private MGOOANL
$  DLENV ! delete environment
$  CRENV SUSI may5 /OFFL ! Create $DBM, $DSP, $ANL
                         ! Use MAY5.EXE for analysis
$  DLENV ! delete environment
$  CRENV SUSI may5 /DELE ! Create $ANL
                         ! Use MAY5.EXE for analysis
                         ! delete old log files
```

CTRL_T

CTRL_T [process][output]

PURPOSE	Similar to an interactive CTRL_T, but works in DCL procedures.
ARGUMENTS	
process	optional process name
output	optional output (def=SYS$OUTPUT)

Description

FUNCTION	Process information of specified or current process is written to output (SYS$OUTPUT).
Version	1.01
Author	H.G.Essel
Last Update	28-JUL-1986

DEBUG_WINDOW

DEBUG_WINDOW window
DEBWIN window

PURPOSE Setup windows and inits for debug.

ARGUMENTS

window Desired debug window style:

- **DECW or MOTIF** A setup is generated to use standard DECwindow/Motif debugger windows. If you are not on a workstation screen, the standard screen initialisation is enabled.
- **VWS or none** On VWS or DECwindow workstations a separate debug window will be opened for the conventional debug in/output. The standard screen initialisation is enabled.
- **terminal** When a terminal is specified, the debug input and output is directed to this terminal. The standard screen initialisation is enabled.

Description

FUNCTION -

REMARKS -

EXAMPLE

```
$ DEBWIN MOTIF
```

DLENVIR

DLENV*IR

PURPOSE	Delete current environment and all subprocesses
FUNCTION	1.Defines logical LNM$GOOSY_TABLES to LNM$GOOSY. 2.Deletes all GOOSY components. 3.Deletes the environment. 4.Sets process name to Y_env_#.
Version	1.01
Author	H.G.essel
Last Update	14-APR-1987

DTENVIR

DTENV*IR

PURPOSE	Detach environment.
FUNCTION	Deassigne logical GOOSY_PROMPT. Resets prompter GOOSY to MGOOTP0
Version	1.01
Author	H.G.essel
Last Update	6-Feb-1990

ETHDEF

ETHDEF destination ethernet protocol interface

PURPOSE	Define logicals for ethernet connection to VME.
ARGUMENTS	
destination	Destination node (E5ELXA). Valid values: E5ELXA, E5ELXB
ethernet	Interface type. Valid values: Microvaxes:QB V8600A,V8600B:UB V6000a,V780a:BI 4000-90:WZ other workstations: WS
protocol	Protocol type. Valid values: TMR
interface	Parallel interface. Valid values: UUA0:

Description

FUNCTION	Defines logical names for ethernet connection to NET processor in VME.
Version	1.01
Author	H.G.Essel
Last Update	25-APR-1990

GOOCONTROL

GOOC*ONTROL [CREATE]or [DISMOUNT]

PURPOSE Defines logical name GOO$CONTROL for control data base. The data base is created and mounted optionally.

ARGUMENTS

CREATE A new data base is created. The name is composed of GOOCTRL_group_node.SEC

DISMOUNT The data base is dismounted. This should be done during LOGOUT.

Description

CALLING GOOC*ONTROL [CREATE]

ARGUMENTS

CREATE A new data base is created

FUNCTION The programs MGOOTMR and MGOOANL keep their control information in a database GOO$CONTROL, if it exists. The command SHOW GOOSY STATUS may be used to display this information on screen. The command GOOCONTROL creates such a data base and defines the logical name GOO$CONTROL. The "physical" data base name is GOOCTRL_<group>_<node>, where node is the one letter node name, group the hex group number. The command should be included in the LOGIN procedure.

REMARKS On each node there must be a control data base. If the section file does not exist, it is created. If the data base is not mounted, this is done.

EXAMPLE GOOCON CRE
creates a data base GOOCTRL_131_E,
if called on EMMA from GOOFY (Group 203).

GUIDE

GUIDE facility level /INIT=string/BRIEF/LIST/LASER

PURPOSE Menu driven guide to use facilities.

ARGUMENTS

facility The name of the facility to be used, e.g. GOOSY If ommitted or asterisk, all available facilities are listed. Then a facility is prompted.

level An optional level specification to enter a specific menu level. The specification is in the form:
n1.n2.n3...
If an asterisk is given, all available levels are listed. Then a level is prompted.

/INIT=string This string will be passed to the initialization procedure as one argument.

/BRIEF Some command procedures display information about the actions to be done. Parts of this output can be suppressed by /BRIEF.

/LIST All menu levels are displayed together with their file names. Note that the 'real' filenames are prefixed by GU_facility_.

/LASER For file output no highlight mode is written. The LN03 is capable to print highlight. To include highlightening in the output file, use /LASER All list output may be directed to a file by
GUIDE/OUTPUT=file ...

Function

FUNCTION A menu driven guide is entered. The menus show topics marked by numbers. If a number is followed by a hyphen(-) the topic points to another menu. If not, it executes a command procedure. The top line shows with "path =" the topic numbers to reach the current menu. It shows with "last topic =" the previous topic after a return.

Next menu

Entering a number the menu of the selected topc is displayed or the command procedure is executed, respectively.

Enter number to select topic : 2 (select topic 2)

Exit a menu

The previous menu is entered by typing "B" or "b" or <CTRL>Z or just <RETURN>. The guide is left by "E" or "e" or <CTRL>Y.

Enter number to select topic : e (exit GUIDE)

Enter number to select topic : B (prev. menu)

Select menu

To enter another menu directly, one can specify a level expression by "=n1.n2....". Then GUIDE enters the menu n1.n2.n3 (counting from the beginning). This is the same as leaving GUIDE and calling again with the level specification.

Enter number to select topic : =1.2.1 (select menu 1.2.1)

Enter number to select topic : = (select top menu)

One can jump over menus by input of a level expression "n1.n2.n3". Then these topics are selected beginning from the current one.

Enter number to select topic : 2.4 (select topic 2.4)

DCL proc.

A DCL command line can be entered behind an at (@). The command will be executed in a spawned subprocess. Note that the @ means NOT to execute a DCL procedure! To do that, two @'s are necessary:

Enter number to select topic : @DIR *.PPL

Enter number to select topic : @@dclproc

HELP

DCL help can be invoced directly by:

Enter number to select topic : H keywords

Examples

EXAMPLE

```
GUIDE ! display facilities
  GUIDE GOOSY ! enter first level menu
  GUIDE GOOSY 1.2 ! enter menu 1.2
  GUIDE/OUTPUT=GOOSY_GUIDE.LIS GOOSY /LIST
                          ! write all levels into file
  GUIDE GOOSY * ! display all levels and
                          ! prompt for level
```

Examples for answering the prompt:

Enter number to select topic : 2 (select topic 2)

Enter number to select topic : 2.4 (select topic 2.4)
starting at current level.

Enter number to select topic : =1.2.1 (select menu 1.2.1)
starting from level 0.

```
Enter number to select topic : = (select top menu)
Enter number to select topic : e (exit GUIDE)
Enter number to select topic : @DIR *.PPL
                               execute command DIR *.PPL in spawn.
Enter number to select topic : @@XYZ
                               execute DCL procedure XYZ
Enter number to select topic : ? (HELP GUIDE)
Enter number to select topic : H string (HELP)
```

Guide-Programming

Function

GUIDE processes two kinds of files:

1. Text files with type .TXT
2. DCL procedures with type .COM

The file names are:

GU_facility_menu.TXT or GU_facility_menu.COM

<facility> is specified by calling GUIDE,

<menu> is specifed in the text files for the menu levels.

The first level file must be GU_facility.TXT

The text files contain the menu information. All text and DCL files must reside on the same directory. This directory is translated from logical name GUIDE$facility. If this logical name is not defined, the files are looked up from the directory of GUIDE. The format is described in the following:

Menu-Design

The menus ore defined in text files

GU_facility_menu.TXT

An exclamation mark (!) at the beginning of a line marks comments. These lines are ignored as well as empty lines.

The first line must be the menu headline preceded by an !. The next line must be empty.

At the beginning of the line there must be the menu name for the menu to be called by that line. If the next level is no text, but rather a DCL procedure to be executed, the menu name must be preceded by an "@".

Behind two bars (——) the text to be displayed follows. This text is used as headline for that topic menu. If no bars are found, the line is displayed as continuation line (double bars are not allowed). Example:

```
! comment line
XYZ—— menu line (enters next menu)
@ABCDE—— menu line (executes DCL procedure)
        continuation without double bars
```

Here, topic 1 enters the next menu level reading text from file GU_facility_XYZ.TXT

Topic 2 executes a DCL procedure named GU_facility_ABCDE.COM The text files should not contain more than 19 true text lines (without comments).

Command-procedures

The command procedures executed by GUIDE should handle CONTROL_Y and ERRORs in a proper way. They should report at the end a success or error. When they display information on the screen they should prompt for a <RET> to continue to give the user the chance to read the output. Prompts from the terminal should be done by

```
$ READ/END=G_cont/PROMPT="string" SYS$COMMAND line
$ G_cont:
```

This avoids the answers to be written in the terminal recall buffer. The END label is reached by CONTROL_Z. Note that the symbol 'line' is NOT changed in that case. Another way is to use the GUIDE_PROMPT procedure. The symbol PROMPT is defined in GUIDE.COM:

```
$ PROMPT "string" "default"
```

The answer is in global symbol PROMPT_ANSWER.

If the prompt was broken by <CTRL>Z , a 3 is returned in $STATUS and PROMPT_ANSWER is "". The default specification is optional. There can be optionally specified a HELP keyword as P4. Then a ? as input enters HELP with that keyword. The guide procedures may also be call MDCLLIST to get parameters. The procedures are always called with a ? as P1. Then MDCLLIST enters a parameter menu.

GUIDE sets a global symbol GUIDE_VERB (verbosity).

GUIDE_VERB is TRUE by default.

GUIDE_VERB is FALSE if GUIDE is called with /BRIEF.

Using that symbol one can control the verbosity of output.

GUIDE-initialization

Specific initializations for a guide can be done in a command procedure named

```
GUIDE$facil:GU_facil_INITIALIZATION
```

This procedure is called before any menu is entered. It is not called for listings (/LIST/FILE/MENU).

GUIDE-finish

Specific finish actions for a guide can be done in a command procedure named
GUIDE$facil:GU_facil_FINISH
This procedure is called before leaving guide.
It is not called for listings (/LIST/FILE/MENU).

Guide-guide

There is a guide to write guides. This guide is invoked by
GUIDE GUIDE
The files GU_GUIDE* can be used as example how to
write guide files.

Qualifiers

CALLING	GUIDE facility level /BRIEF/LIST/LASER /FULL/FILES/MENU
/FULL	All sources are included in the output.
/FILES	Outputs a list of all used files.
/MENU	Outputs all menus.

LANL

LA*NL obj_list /OLB=objlib/OPT=optfile/CMD=cmdfile
/EXE=exefile /MAP=mapfile
/DEBUG /SHARE/NOSHARE

PURPOSE Link user specific analysis program

ARGUMENTS

obj_list List of user modules to be linked. Default unpack routines are provided for J11 data and compressed data (analysis output). For these input data the default analysis program J11 can be used, if no user analysis routine is needed.

/OLB=objlib optional user object library

/OPT=optfile optional link options file

/CMD=cmdfile File containing one line of link qualifiers. This name is appended to the link command by @cmdfile which means that the line is inserted in the command line before the command is excuted.
NOT VALID with /DEB switch.

/DEBUG The debugger is linked.
NOT VALID with /CMD switch.

/SHARE Valid together with /DEB switch. If not specified, image is linked from object libraries. If specified, image is linked from sharable images as it is without /DEBUG switch.

/NOSHARE Link from object libraries.
NOT VALID with /CMD switch.

/EXE=exefile Optional name of the exe file. Default is MGOOANL

/MAP=mapfile Optional name of map file.

EXAMPLES

```
LANL X$EVENT,X$ANAL /DEB
  LANL X$EVENT,X$ANAL /DEB/SHARE
  LANL X$E1,X$A1 /EXE=MYANL1
  LANL * /EXE=MYANL1
  LANL X$J11_ANAL /EXE=MYANL1
```

Description

FUNCTION

The user analysis program is linked. The file name is MGOOANL.EXE if not otherwise specified. The first argument is a list of files to be linked with MGOOANL. If /DEB is specified, but not /SHARE, MGOOANL is linked to object libraries instead of sharable images. This is the reason for longer linking time.

Version 1.01

Author H.G.Essel

Last Update 15-JAN-1987

LINKJ11

LINKJ11 objfile /COMPILE

PURPOSE	Link a J11 stand alone task
ARGUMENTS	
objfile	OBJ filename, created by COMPILE file.MAC. Must be on current directory.
/COMPILE	Optional compile file first.

Description

FUNCTION	-
Version	1.01
Author	H. Grein
Last Update	30-APR-1987

LSHARIM

```
LSHARIM module image
        /GLOBAL=list
        /SHARE*LOG=name
        /MAP=mapfile
        /KEEP
        /GROUP
        /DEBUG
```

PURPOSE Link modules into a sharable image.

ARGUMENTS

module Modules to be linked into sharable image:
1. List of file names (wildcards)
2. @file contains file names
3. @library(module name list)

image Name of generated executable sharable image

/GLOBAL=list Globals which occur in the modules

/SHARE*LOG=n Logical name of the sharable image

/MAP=mapfile Optional map file.

/KEEP Keep all temporary files

/GROUP The logical name for the sharable image is entered in the GROUP table instead of the JOB table.

/DEBUG Link with debugger.

DESCRIPTION

CALLING

```
LSHARIM module image
        /GLOBAL=list
        /SHARE*LOG=name
```

/MAP=mapfile
/KEEP
/DEBUG

ARGUMENTS

module Modules to be linked into the sharable image. The following inputs are possible:

1.) A VAX/VMS file specification list with any wildcards; e.g. X$*USER*. The default file type is .OBJ. The specified modules have to be compiled.

2.) A file, which contains the list of modules to be linked into the sharable image. The file has to be specified with a leading "@"; e.g.

@list.dat

Per line one module name is expected. The modules have to be compiled.

3.) A library specification; e.g

@opriv.olb(X$*)
@opriv.olb(X$USER_ANAL)

image Name of generated executable sharable image file. Default type is .EXE.

/GLOBAL=list All FORTRAN COMMON blocks or PL/I EXTERNAL variables have to be placed in unshared sections. Therefore all globals occuring in your program have to be known. If only one global global parameter occurs define it directly:

/GLOBAL=name

Futhermore a file containing a list of all used global paramters can be specified,e.g.:

/GLOBAL=@list.dat

In each line one global parameter name is assumed.

/SHARE*LOG=n Logical name assigned to the the sharable image in JOB table. If not specified, the sharable image name is used.

/MAP=mapfile Optional map file.

/GROUP The logical name for the sharable image is entered in the GROUP table instead of the JOB table. You need GRPNAM priviledge for that. Note that the the logical name is valid for all sessions in the group.

/KEEP Keep all temporary files

/DEBUG Link with debugger.

FUNCTION

One or several modules can be linked together into a sharable image. The global parameters refered in the modules can be specified and are placed in unshared sections of the sharable image. For control a linker map file is genereated.

The sharable image can be referenced only by the logical name. This name is defined only during the session or untill next system startup (/GROUP). Therefore one should add the definition in the LOGIN.COM with /JOB qualifier.

Example

LSHARIM x$*.obJ anal.exe /MAP=anal.map/share=usershr

All modules starting with X$ are linked into the generated sharable image "ANAL.EXE". The map file "ANAL.MAP" will be produced and the logical name "USERSHR" is assigned to the sharable image.

LSHARIM x$start,x$stop anal.exe

Modules x$start.obj and x$stop.obj are used building the sharable image "ANAL.EXE".

LSHARIM @file.dat anal.exe

All modules listed in "FILE.DAT" are used building the sharable image "ANAL.EXE".

LSHARIM @opriv(X$*) anal.exe

All modules X$* foung in the obejct library opriv are linked to a sharable image.

MANUAL

MANUAL PRINT
/INTRO/DISPLAY/ANALYSIS/DATABASE/VME/HARDWARE
/BUFFER/VMS/ACQCOM/ANACOM/ALL/ACQUISITION

PURPOSE	Print GOOSY manuals.
ARGUMENTS	
/INTRO	GOOSY introduction
/DISPLAY	GOOSY display including commands
/ANALYSIS	GOOSY data acquisition and analysis including commands and macros
/ACQUISITION	GOOSY data acquisition.
/DATABASE	GOOSY data base manager including commands
/VME	VME frontend system
/HARDWARE	Hardware description (J11 and MBD) and buffer structures
/BUFFER	GOOSY buffer and event structures
/VMS	GSI introduction to VMS
/ACQCOM	GOOSY transport manager commands
/ANACOM	GOOSY analysis manager commands and macros
/ALL	All manuals (Caution: A lot of paper!)

DESCRIPTION

CALLING	MANUAL PRINT /INTRO/DISPLAY/ANALYSIS/DATABASE/VME/HARDWARE /BUFFER/VMS/ACQCOM/ANACOM/ALL/ACQUISITION
ARGUMENTS	

FUNCTION Print specified manuals doublesided on postscript printer in computer center printer room.

MTAPE

MTAPE device name
/INI*TIALIZE/DENS*ITY=d/BLOCK*SIZE=b/DIS*MOUNT

PURPOSE	Initialize and mount a GOOSY tape
ARGUMENTS	
device	Logical name of tape unit.
name	Label name of the tape
/INITIALIZE	Initialize tape
/DENSITY=d	Tape density, default=6250
/BLOCKSIZE=b	Blocksize in Kbyte, default=24. Should be multiple of GOOSY blocksize. Default is normally adequate.
/DISMOUNT	A tape already mounted is dismounted first. You must mount the new tape on the device and hit <RETURN> to continue.

Description

FUNCTION

The tape is optionally initialized and then mounted. The density specification is used for initialization, the blocksize for mounting.

If /DISMOUNT is specified, the tape presently mounted (if any) is dismounted. Then You must mount the new tape on the device, and enter <RETURN> to continue. Without the /DISMOUNT qualifier it is assumed that the desired tape volume is already mounted on the device.

Version	1.01
Author	H.G.Essel
Last Update	8-OCT-1987

OPSER

OPSER command

PURPOSE Execute priviledged operator commands

ARGUMENTS

command ? -> enter main menu
SEARCH (GOOSY)
DIFFER (GOOSY)
COMPILE (GOOSY)
REPLY
TAPE
SET

Description

FUNCTION The commands supported by OPSER are executed by an operator server.

EXAMPLE To enter main menu:
$ OPSER
To enter sub menu for REPLY:
$ OPSER REPLY

PLOTMET

PLOTMET metafile type command plotter
/COPIES=c /FONT=f

PURPOSE Plot a metafile on specified plotter

ARGUMENTS

metafile Name of the metafile which should be plotted on the specified queue or physical device.

type Device type for format. The following types are supported by GOOSY:

1.) LN03 Laser printer (=default)
2.) HP7550A3,HP7550A4 pen plotter
3.) POST postscript

command Optional print command (enclose in ""). If specified, plotter is ignored.

plotter Queue name or physical address of the plotter. If a colon (":") is specified at this position it is assumed that a physical adress has beeen specified. Is ignored when a print command is given.

copies Number of copies which should be printed. If no Printer queue is specified "copies" is ignored. (default=1)

font Font to be used to modify the default text bundle table. This argument could be used to produce pictures with nicer lettering. (default=0)

Description

FUNCTION This procedure plots the specified metafile on a plotter.

NOTE One may format to POSTscript format, but print on LN03 printers. In this case use the command "P x POST" where x is A,B,C...

Example

```
PLOTMET x.meta POST "PS A POST"
  PLOTMET x.meta POST "P A POST"
  PLOTMET x.meta LN3 "" SYS$LN03_A
```

File name	PLOTMET.COM
Dataset	-
Version	1.01
Author	W. Spreng
Last Update	19-NOV-1986

SELECT_MBD

SELECT_MBD mbd

PURPOSE Select a valid MBD controller on a VAX

ARGUMENTS

mbd I The device code of a MBD (A or B)
A : Normally the only or the first MBD on a VAX.
This is the only one for micro-VAXes.
B : The second MBD on VAX-8600 (DONALD or EMMA).

Description

CALLING SELECT_MBD mbd

ARGUMENTS

mbd I The device code of a MBD (A or B)
A : Normally the only or the first MBD on a VAX.
This is the only one for micro-VAXes.
B : The second MBD on VAX-8600 (DONALD or EMMA).

FUNCTION The command procedure will check any existing MBD, the VAX where the selection is done, and it will define the logical names:
MBDA, MBDB, and MBDC
in the job logical name table of the caller.
The GOOLOG command procedure will define the logical names to be invalid to make shure that the user will call this procedure before he can access any CAMAC function.

REMARKS Must be called once before any CAMAC function can be performed.

EXAMPLE SELECT_MBD B ! Selects the 2nd MBD

Utility UTIL

Home direct. GOO$EXE

File name	SELECT_MBD.COM
Author	M. Richter
Created	15-FEB-1988
Last Update	19-FEB-1988

SETMESSAGE

SETMESSAGE facility qualifier

PURPOSE	Control Message output of GOOSY and VMS
ARGUMENTS	
facility	GOOSY: GOOSY Messages VMS : VMS Messages
qualifier	see below

Description

FUNCTION	Enables or Dissables different levels of messages
Version	1.01
Author	R.Thomitzek
Last Update	20-OCT-1988

GOOSY

CALLING	SETMESSAGE GOOSY /NOHEADER/NOPREFIX/LAST/ON/OFF/SHOW
/NOHEADER	Message Headerline of GOOSY-Messages will be suppressed
/NOPREFIX	Message Prefix of GOOSY-Messages will be suppressed
/LAST	Only last Message will be output
/SHOW	Show message setting
/ON	Full message
/OFF	Same as /NOHEADER/NOPREFIX

Example

```
SETM GOOSY /NOH ! no header
  SETM GOOSY /ON ! full message output
  SETM GOOSY ! full message output
  SETM GOOSY /SHO ! Show message output setting
  SETM GOOSY /OFF ! no header, no prefix
```

VMS

CALLING SETMESSAGE VMS /ON/NOPREFIX

/ON Switch ON all output of VMS Messages

/NOPREFIX Switch OFF all parts of VMS Messages except text.

Example

```
SETM VMS /NOP ! no facility, severity and id
  SETM VMS /ON ! full message output
  SETM VMS ! full message output
```

TLOCK

TLOCK

PURPOSE Lock terminal by password

Description

FUNCTION Locks terminal by password. The password is prompted after call. Then it must be reentered to leave the procedure. If you forget the password, the job must be canceled!

Version 1.01

Author H.G.Essel

Last Update 1-DEC-1988

VMESTRUC

**VMESTRUC inputfile /PLI/FOR/C/PLIB=/CLIB=/FLIB=
/GLPUT/DELETE**

PURPOSE	Generate declarations from language independent source.
ARGUMENTS	
inputfile	File containing language independent decla- rations. Specify library module as library(module).
/PLI/FOR/C	Controls, which output is generated.
/PLIB/FLIB=	Optional VMS text libraries to store the generated modules.
/CLIB=	Optional directory to store generated modules. I.e. VME$INC:
/GLPUT	Use GLPUT command instead of LIB/REP or COPY
/DELETE	Delete generated files.

Description

CALLING	VMESTRUC inputfile /PLI/FOR/C/PLIB=/CLIB=/FLIB= /GLPUT/DELETE
ARGUMENTS	
inputfile	File containing language independent declarations. Default file type is .VMES. Specify library module as library(module).
/PLI	Output PL/1 include file. Filename is inputfile.PINC.
/FOR	Output FORTARN include file. Filename is inputfile.FINC.
/C	Output C include file. Filename is inputfile..
/PLIB/FLIB=	Optional VMS text libraries to store the generated modules witrh PL/1 or FORTRAN syntax.

/CLIB=	Optional directory to store generated modules with C syntax, i.e. VME$INC:
/GLPUT	Use GLPUT command instead of LIB/REP or COPY
/DELETE	Delete generated files.
FUNCTION	Declarations of constants, variables and structures are translated into the proper PL/1, FORTRAN or C statements. The syntax of the source file is described below. If none of the qualifier is selected, all three output files are generated. The syntax of each include file is checked by a test compilation.
EXAMPLE	VMESTR sysctrl generates sysctr.pinc, sysctrl.finc and sysctrl..

Syntax

The syntax of the source file is:
! comment at any place
DEFINE constant value
PREFIX letter
[EXTERNAL]LONG [POINTER]name[(i1,i2)]
[EXTERNAL]WORD [POINTER]name[(i1,i2)]
[EXTERNAL]BYTE [POINTER]name[(i1,i2)]
[EXTERNAL]BIT [POINTER]name[(i1,i2)]
[EXTERNAL]FLOAT [POINTER]name[(i1,i2)]
[EXTERNAL]STRUCTURE [POINTER]structure name
the structure must be known.
STRUCTURE [POINTER]structure
structure declarations
ENDSTRUCTURE
SWAP
lines to be swapped in order in C output
ENDSWAP
%%P this line for PL/1 only
%%C this line for C only
%%F this line for FORTRAN only

Definition values can be specified in decimal, hex (%X...), octal (%O...) string. Character striong must be enclosed in "" All keywords except POINTER may be abbreviated. The array dimensions may be constants previously defined. The variable names for PL/1 and C are prefixed by type letters, i.e. L_ for longword. Structure members are prefixed by type letter, prefix letter and dollar sign. Structures may be nested up to 2 levels:

```
STRUCTURE POINTER X
  STRUCTURE Y
    LONG Y1
  ENDSTR
ENDSTR
```

EXAMPLE

S1

```
Source:
  prefix A
  str pointer x
       long x_1
  swap
       word x_2
       word x_3
  endswap
  %%P word x(LA$x_1-2)
  %%C word x(1)
  endstr
  %%C extern struct x x1(10)
  %%P extern struct x x1(10)
generates PL/1:
  DCL P_SA$x POINTER ;
  DCL 1 SA$x BASED(P_SA$x),
       2 LA$x_1 BIN FIXED(31),
       2 IA$x_2 BIN FIXED(15),
       2 IA$x_3 BIN FIXED(15),
       2 IA$x(LA$x_1-2) BIN FIXED(15);
  DCL 1 SA$x1(10) LIKE(SA$x) EXTERNAL;
and C:
  struct s_x
  {
  long l_x_1;
  short i_x_3;
  short i_x_2;
  short i_x[1];
  } *p_x;
  extern struct s_x s_x1[10];
```

S2

```
Source:
   prefix N
   %%P long SN$y_1
   str pointer y
        long y_1
   %%P str y_2(L_SN$y_1 REFER(LN$y_1))
   %%P byte y_3
   %%P byte y_4
   %%P byte y_5
   %%P byte y_6
   %%P endstr
   %%C long y_7(1)
   endstr
generates PL/1:
   DCL L_SN$y_1 BIN FIXED(31);
   DCL P_SN$y POINTER ;
   DCL 1 SN$y BASED(P_SN$y),
        2 LN$y_1 BIN FIXED(31),
        2 SN$y_2(L_SN$y_1 REFER(LN$y_1)) ,
        3 HN$y_3 BIN FIXED(7),
        3 HN$y_4 BIN FIXED(7),
        3 HN$y_5 BIN FIXED(7),
        3 HN$y_6 BIN FIXED(7);
and C:
   struct s_y
   {
   long l_y_1;
   long l_y_7[1];
   } *p_y;
```

S3

```
Source:
   prefix N
   str z
        long z_1
        str z_2(10)
             long z_3
```

```
            long z_4
        endstr
    endstr
generates PL/1:
    DCL 1 SN$z ,
        2 LA$z_1 BIN FIXED(31),
        2 SN$z_2(10) ,
        3 LN$z_3 BIN FIXED(31),
        3 LN$z_4 BIN FIXED(31);
and C:
    struct s_z
    {
    long l_z_1;
    struct s_z_2
    {
    long l_z_3;
    long l_z_4;
    } z_2;
    } z;
```

WCLOSE

WCLOSE file

PURPOSE Wait for file to be closed.

ARGUMENTS

file File to be checked.

Description

FUNCTION Trys to open specified file. If the file is locked, it waits and retries, if not the file is closed. This command should be used to wait for an analysis writing an output file after closing the output file by STOP ANAL OUT /CLOSE because pending buffers are written before the file is closed. If in a DCL procedure the analysis would be deleted after the STOP command, the last buffer cannot be written into the file.

EXAMPLE

```
$ GOOSY START ANAL OUT X.LMD
  $ GOOSY START INPUT FILE Y.LMD
  $ MGOOWAIT ...
  $ GOOSY STOP ANAL OUT /CLOSE
  $ WCLOSE X.LMD
  $ GOOSY DELETE PROCESS $ANL
  $ ...
```

Version 1.01

Author H.G.Essel

Last Update 15-FEB-1989

Appendix C

Analysis 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);
  $ATTACH(BASE,db,W);
  IF ^ STS$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) ^ = 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 D

GOOSY Data Formats

D.1 Introduction

D.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.

D.1.2 Buffer Files

If GOOSY buffers are dumped to files, the first buffer may be a file header buffer (see section D.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.

D.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.

D.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.

D.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

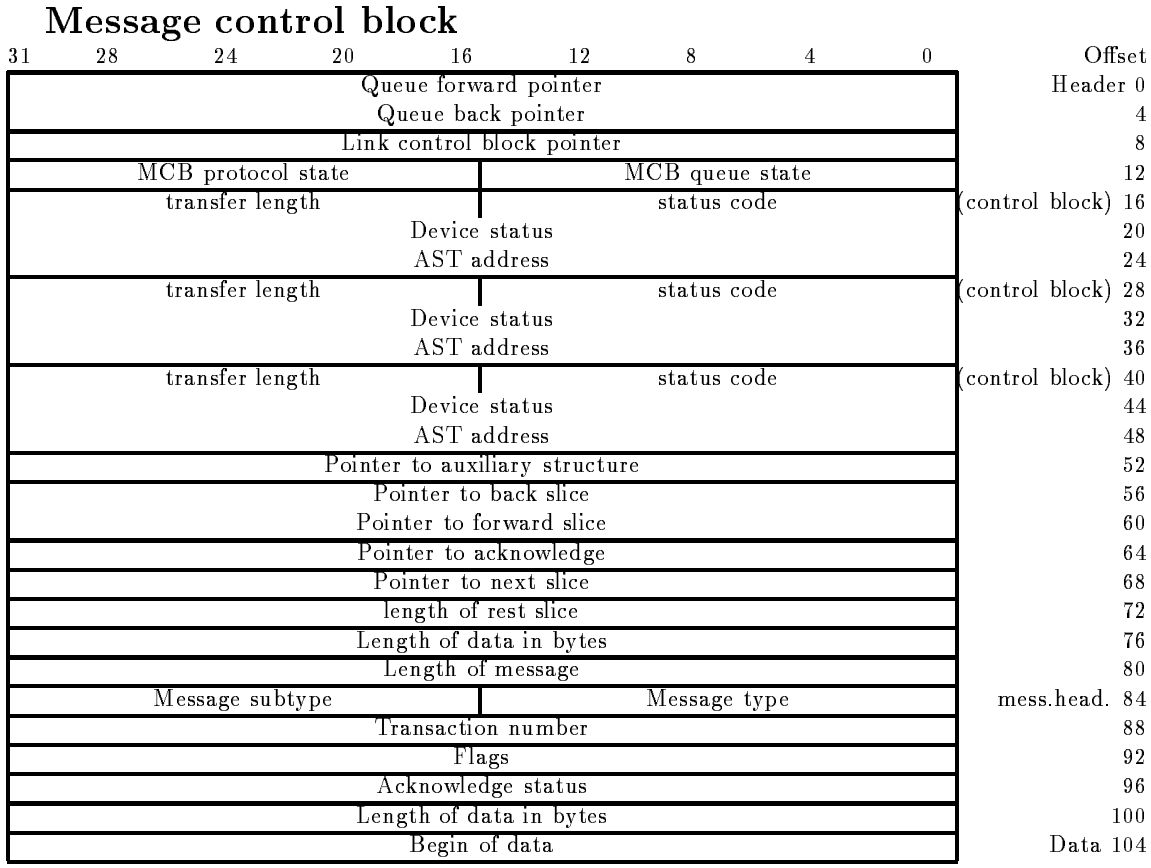


Figure D.1: Message Control Block Structure

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 */
```

D.3 Buffer Structure

D.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 D.3.1) and the event header (section D.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 D.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.

D.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 D.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 D.7.1) and FERA (section D.7.2) event structures, if they are not preprocessed by a frontend processor adding a header.

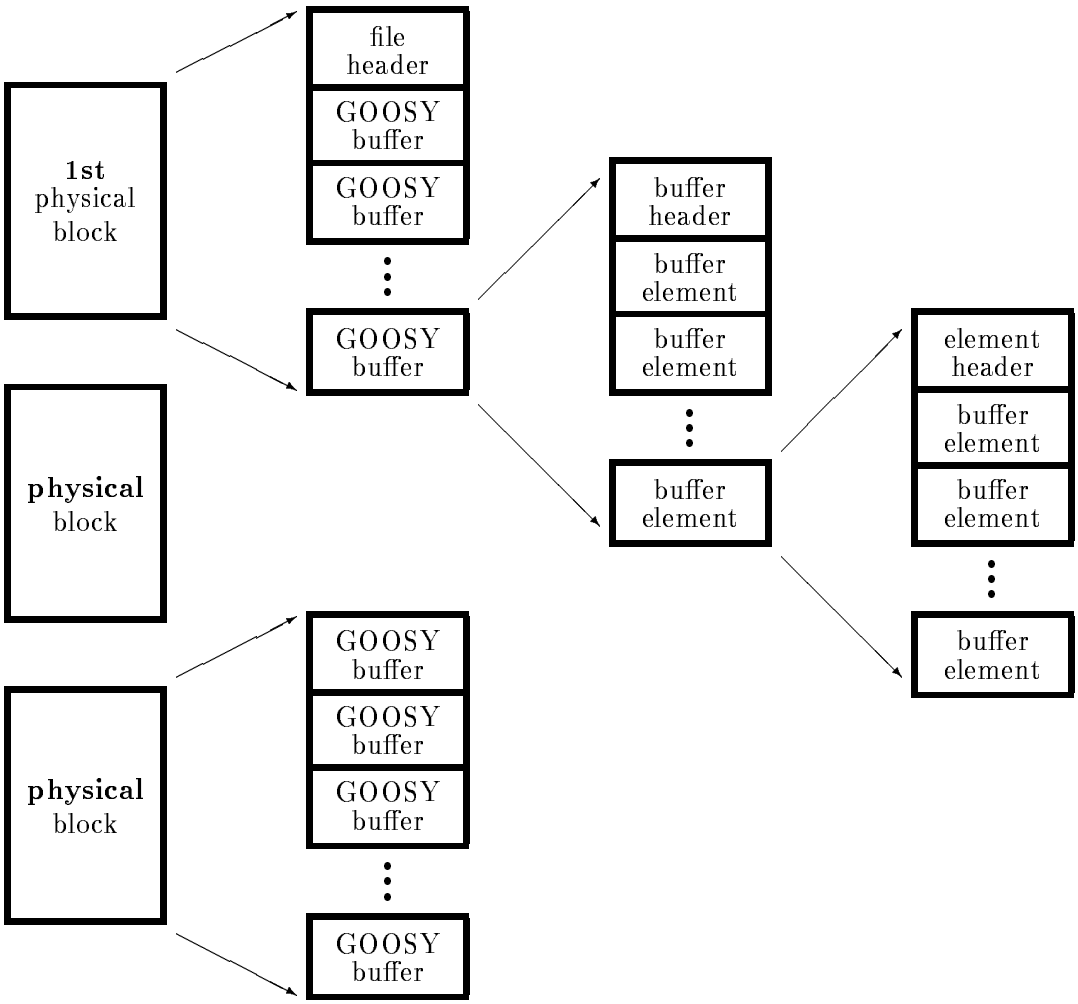


Figure D.2: The GOOSY data structures of a listmode dump file.

D.3.3 Buffer Header

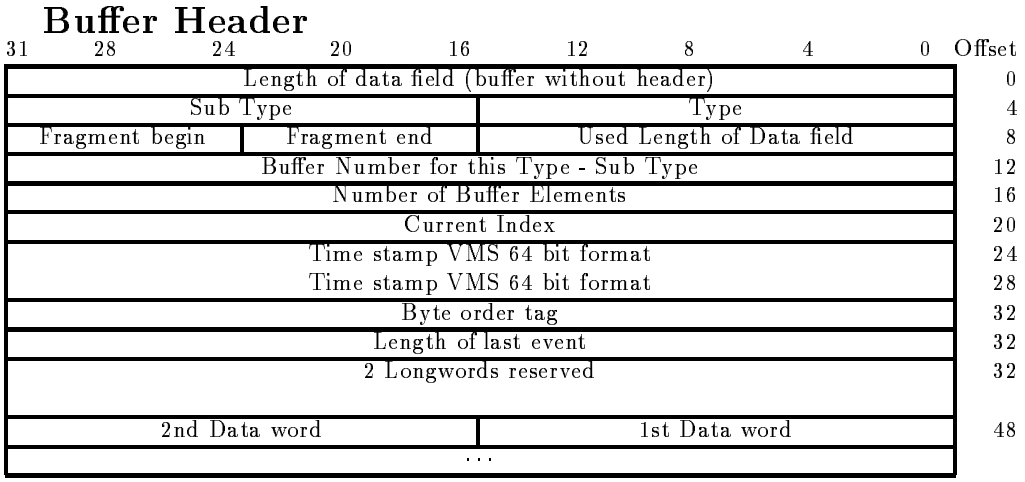


Figure D.3: Buffer Header Structure

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              */
/*---------------------------------------------------------------*/
```

D.3.4 Buffer Element Header

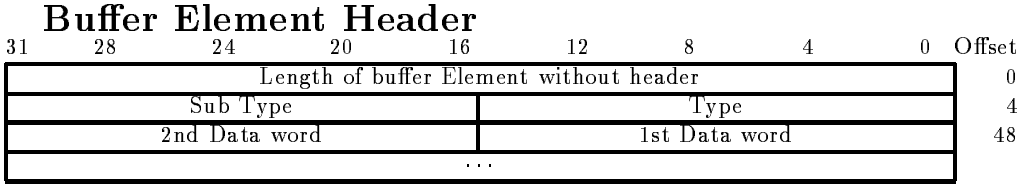


Figure D.4: Buffer Element Header Structure

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     */
/*-----------------------------------------------------------*/
```

D.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.

D.4 Buffer Types

D.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)

D.4.2 File Header Buffer

Figure D.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

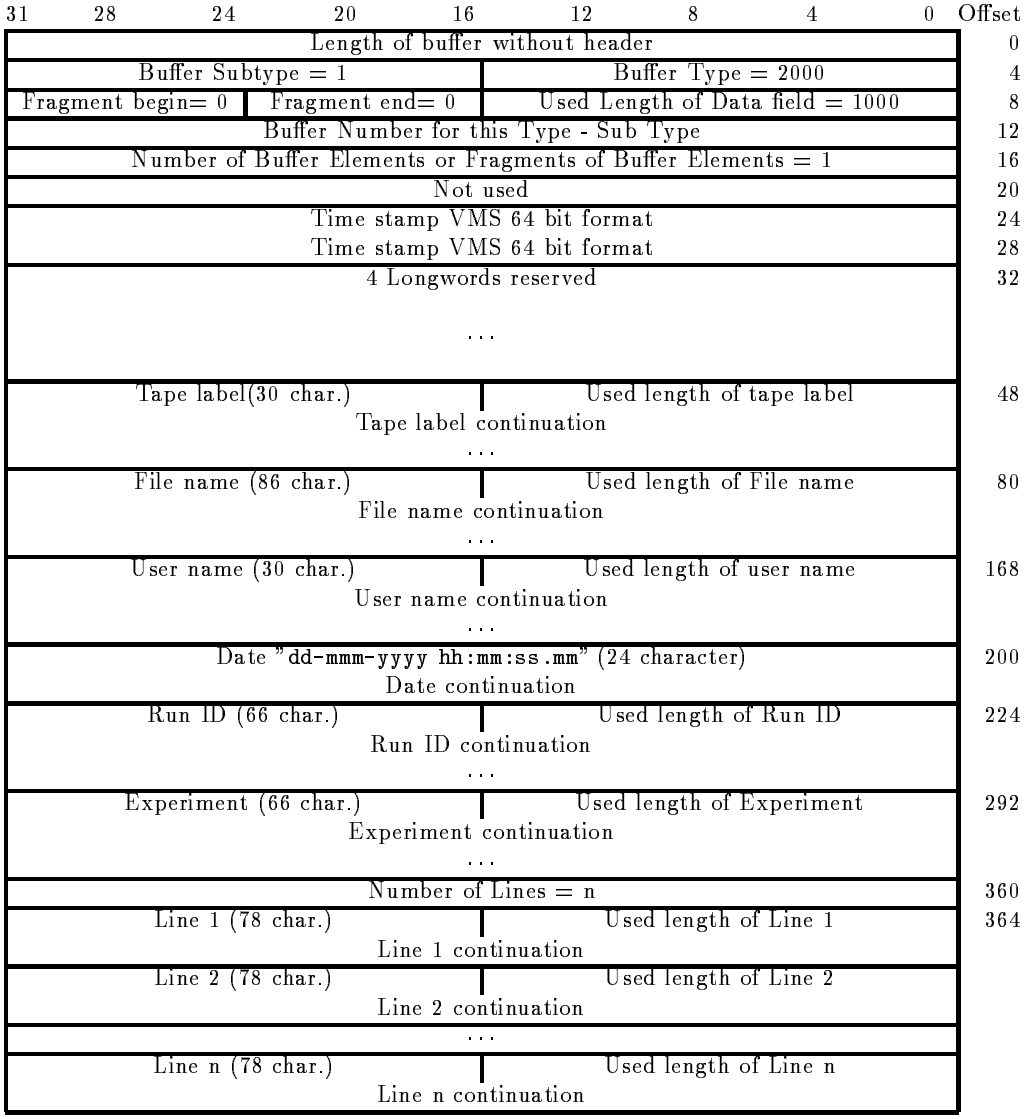


Figure D.5: File Header Structure

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           */
/*--------------------------------------------------------------*/
```

D.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.)

D.4.4 GOOSY Listmode Data Buffers

Listmode data buffers contain buffer elements called *events*. The different event types are described in the next section.

D.5 Event Structures

D.5.1 Event Type 3 (compressed)

Figure D.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!**

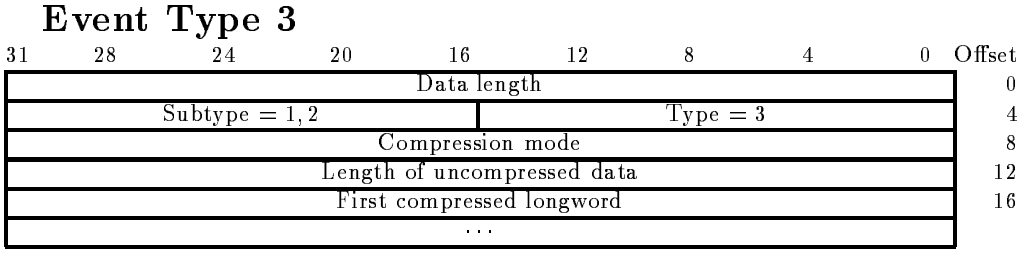


Figure D.6: Event structure type 3 (compressed)

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.

D.5.2 Event Type 4

Event Type 4, Subtype 1 (block)

Figure D.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!**

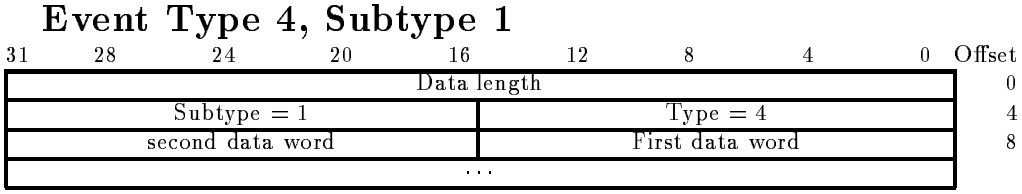


Figure D.7: Event structure type 4, subtype 1 (block)

Event Type 4, Subtype 2 (no zero's)

Figure D.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!**

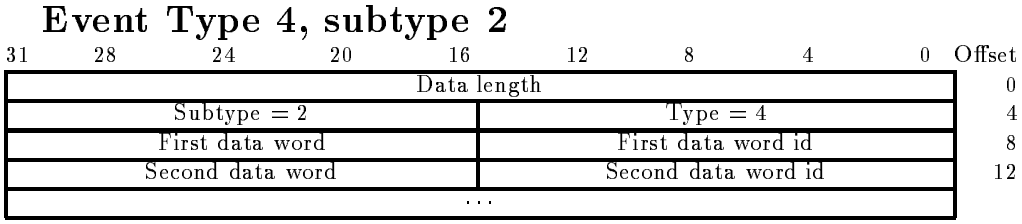


Figure D.8: Event structure type 4, subtype 2 (no zeros's)

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.

D.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 D.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!**

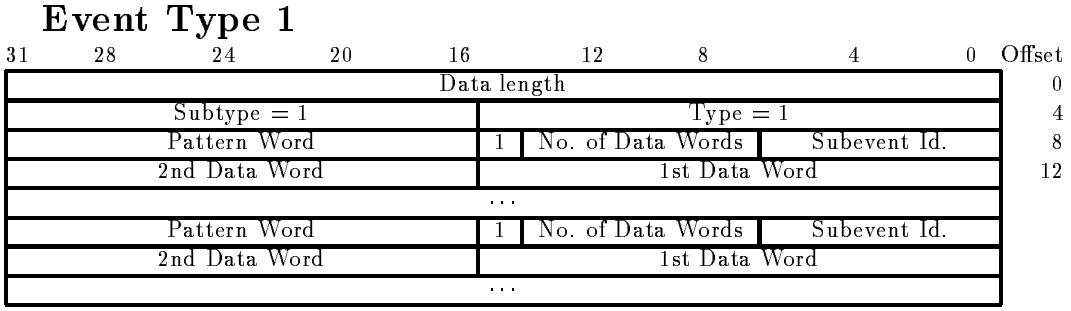


Figure D.9: Event structure type 1 (SILENA)

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.

D.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 D.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)).

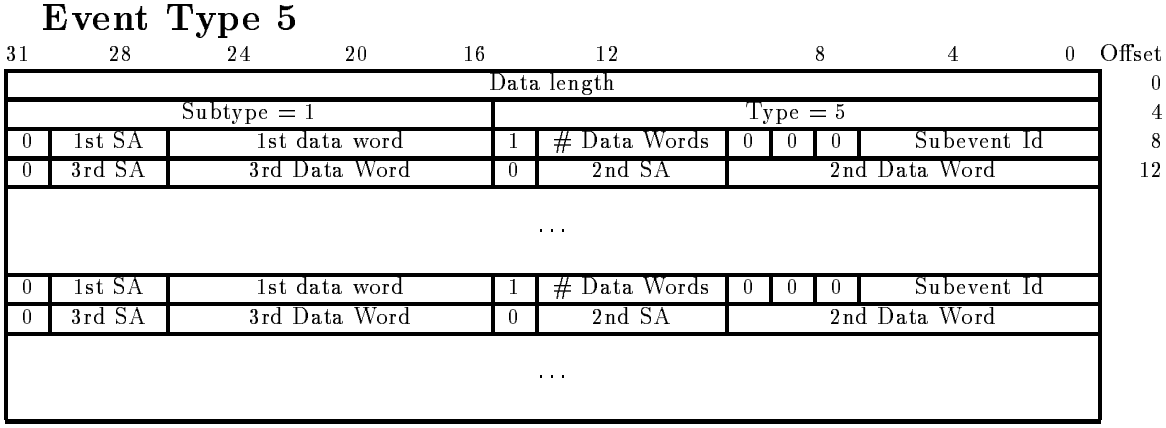


Figure D.10: Event structure type 5 (LRS FERA)

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.

D.5.5 Event Type 6 (MBD buffer type 6)

Figure D.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))

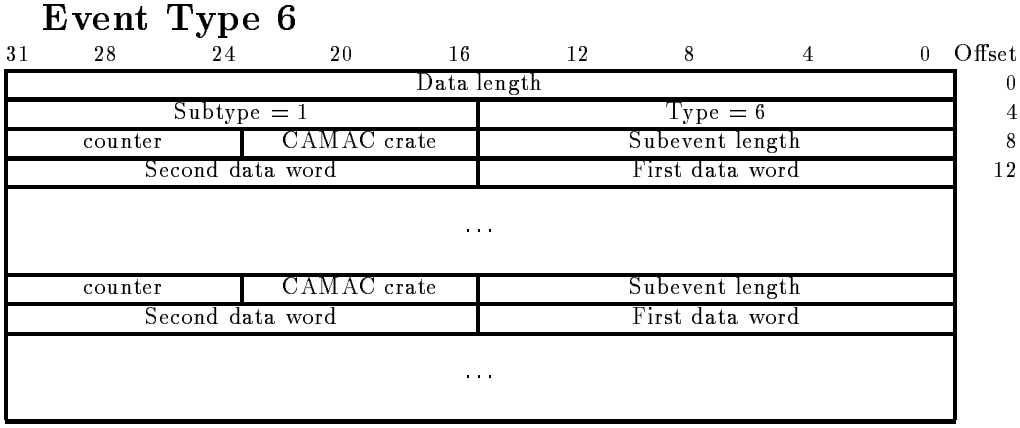


Figure D.11: Event structure type 6 (MBD buffer type 6)

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.

D.5.6 Event Type 7 (MBD buffer type 7)

Figure D.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!**

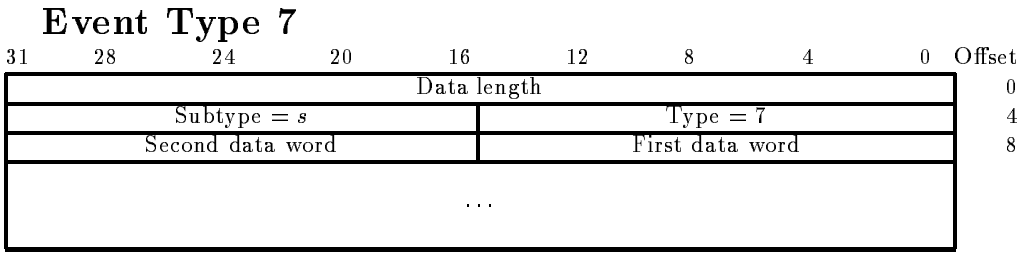


Figure D.12: Event structure type 7 (MBD buffer type 7)

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.

D.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

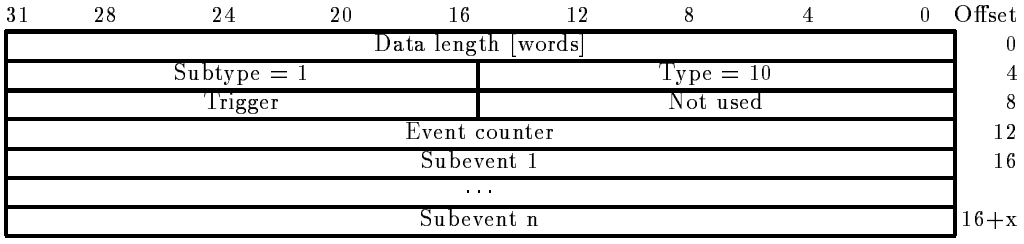


Figure D.13: Event Structure

```
    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

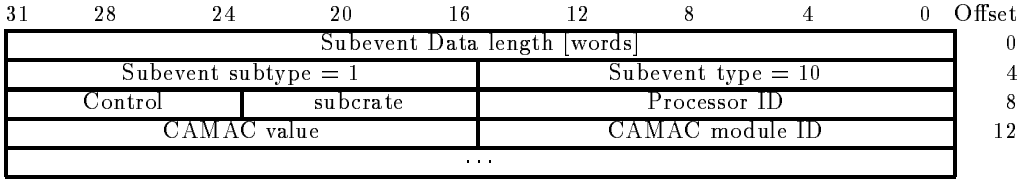


Figure D.14: CAMAC Subevent Structure

```
/* ================= 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.

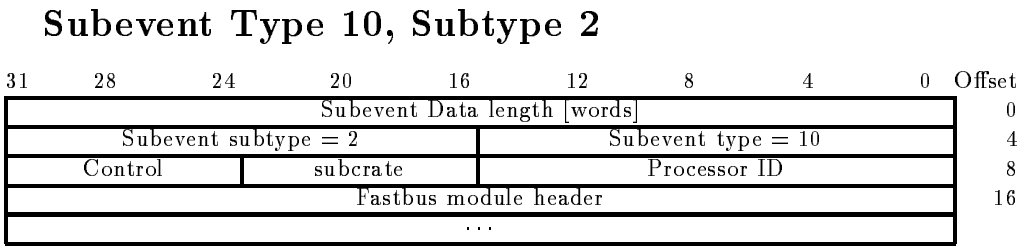


Figure D.15: Fastbus Subevent Structure

The following structure maps to the data field. It is defined in SA$vesfb in library GOOINC.

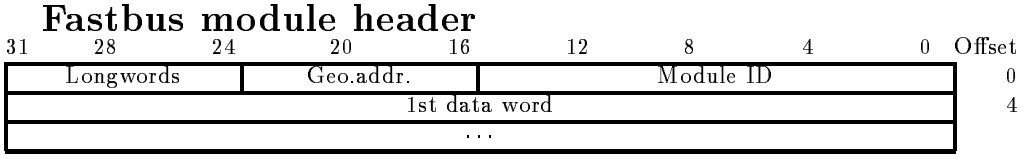


Figure D.16: Fastbus Module header

```
/* 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);
```

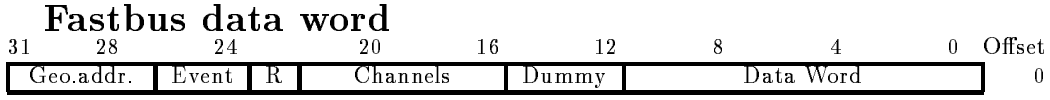


Figure D.17: Fastbus Data Word

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. */
/*-----------------------------------------------------------------------*/
```

D.6 Buffer Element Structures

D.6.1 Buffer Element Type 9000 (Time Stamp)

Figure D.18 shows the Time Stamp structure (buffer element structure type 9000). **These buffer elements are longword aligned!**

Buffer Element Type 9000, Time Stamp

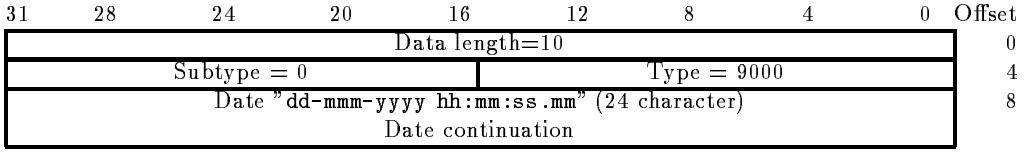


Figure D.18: Buffer Element structure type 9000, Time Stamp

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.

D.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

D.7 Nonstandard Buffer Structures

D.7.1 Buffer Type 12 (SILENA)

Figure D.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!**

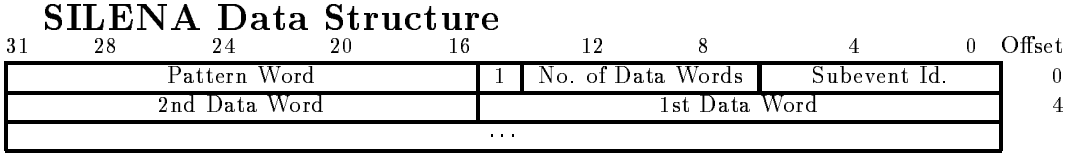


Figure D.19: Data structure SILENA ADC

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.

D.7.2 Buffer Type 15 (LRS FERA)

Figure D.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!**

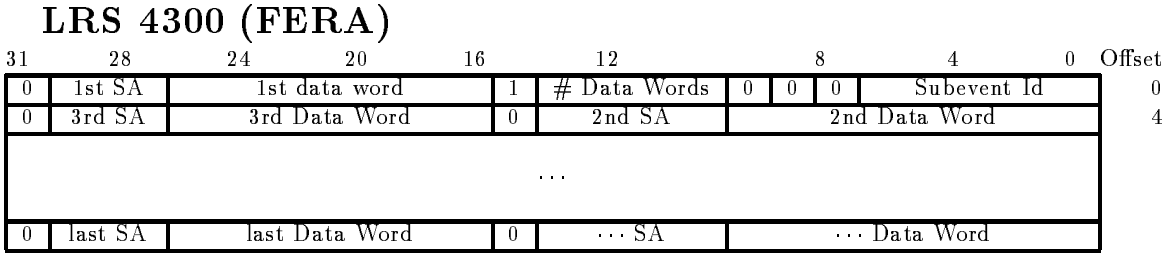


Figure D.20: Data structure LRS FERA

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.

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 **M**icroprogrammed **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

T

U

V

Contents