GOOSY
Id.: BUFFER
Version: 1.0
Date: 14–Jan–1991
Revised: Jan, 14 1991

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

GOOSY Buffer Structures

H.G.Essel et. al.

Jan, 14 1991

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

List of Figures

1.1 Message Control Block Structure 6
1.2 GOOSY File Structure 9
1.3 Buffer Header Structure 10
1.4 Buffer Element Header Structure 12
1.5 File Header Structure 17
1.6 Event structure type 3 (compressed) 21
1.7 Event structure type 4, subtype 1 (block) 22
1.8 Event structure type 4, subtype 2 (no zeros's) 22
1.9 Event structure type 1 (SILENA) 23
1.10 Event structure type 5 (LRS FERA) 25
1.11 Event structure type 6 (MBD buffer type 6) 26
1.12 Event structure type 7 (MBD buffer type 7) 28
1.13 Event Structure 29
1.14 CAMAC Subevent Structure 29
1.15 Fastbus Subevent Structure 30
1.16 Fastbus Module header 30
1.17 Fastbus Data Word 31
1.18 Buffer Element structure type 9000, Time Stamp 32
1.19 Data structure SILENA ADC 33
1.20 Data structure LRS FERA 34

Chapter 1

GOOSY Data Formats

1.1 Introduction

1.1.1 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.

1.1.2 Buffer Files

If GOOSY buffers are dumped to files, the first buffer may be a file header buffer (see section 1.4.2)! 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.

1.1.3 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.

1.1.4 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.

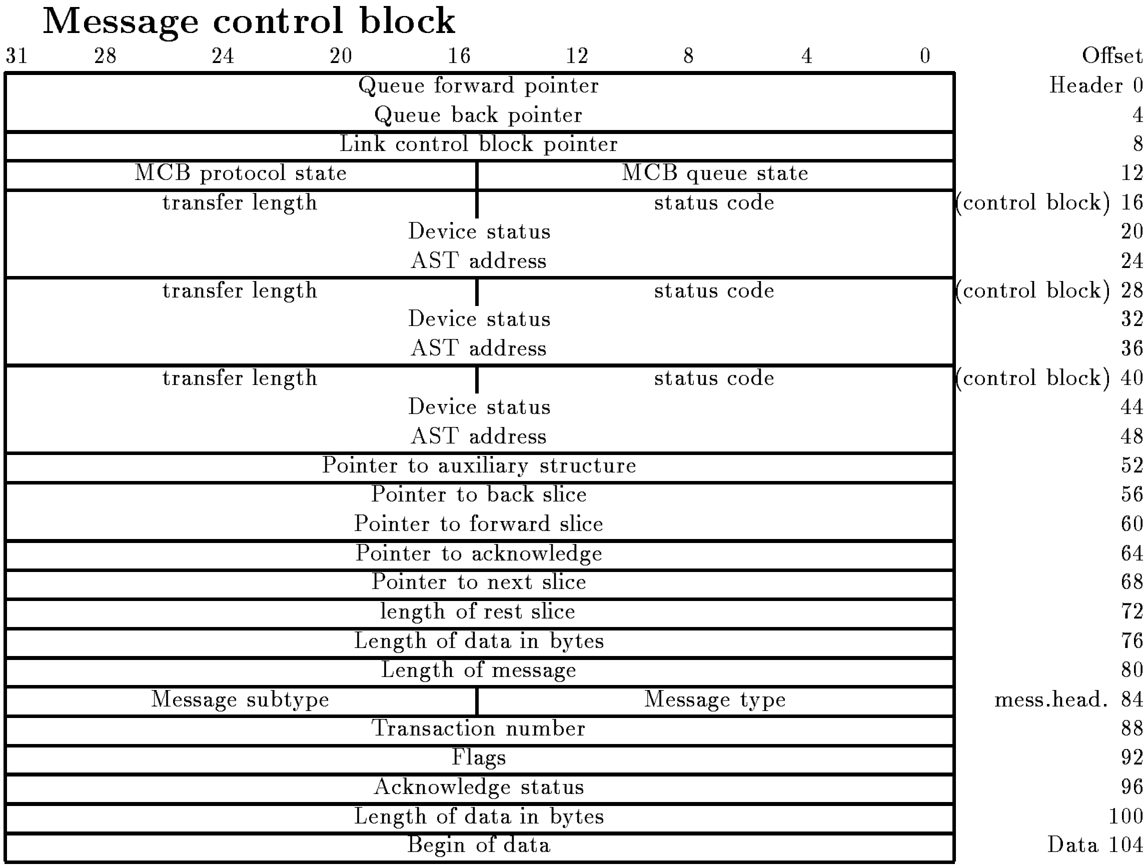
event Data describing one physical event. Events are buffer elements in standard buffers. There are, however, buffers containig events without headers (nonstandard buffers). Events may be composed of subevents.

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.

1.2 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 information is transferred. The message header contains information which is transferred. The structure is found in GOOINC(SN$MCB):



Message control block

31 28 24 20 16	12 8 4 0	Offset
Queue forward pointer		Header 0
Queue back pointer		4
Link control block pointer		8
MCB protocol state	MCB queue state	12
transfer length	status code	(control block) 16
Device status		20
AST address		24
transfer length	status code	(control block) 28
Device status		32
AST address		36
transfer length	status code	(control block) 40
Device status		44
AST address		48
Pointer to auxiliary structure		52
Pointer to back slice		56
Pointer to forward slice		60
Pointer to acknowledge		64
Pointer to next slice		68
length of rest slice		72
Length of data in bytes		76
Length of message		80
Message subtype	Message type	mess.head. 84
Transaction number		88
Flags		92
Acknowledge status		96
Length of data in bytes		100
Begin of data		Data 104

Figure 1.1: Message Control Block Structure

```
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 */
```

1.3 Buffer Structure

1.3.1 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 1.3.1) and the event header (section 1.3.4). 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 1.2 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.

1.3.2 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 1.5.3, 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 1.7.1) and FERA (section 1.7.2) event structures, if they are not preprocessed by a frontend processor adding a header.

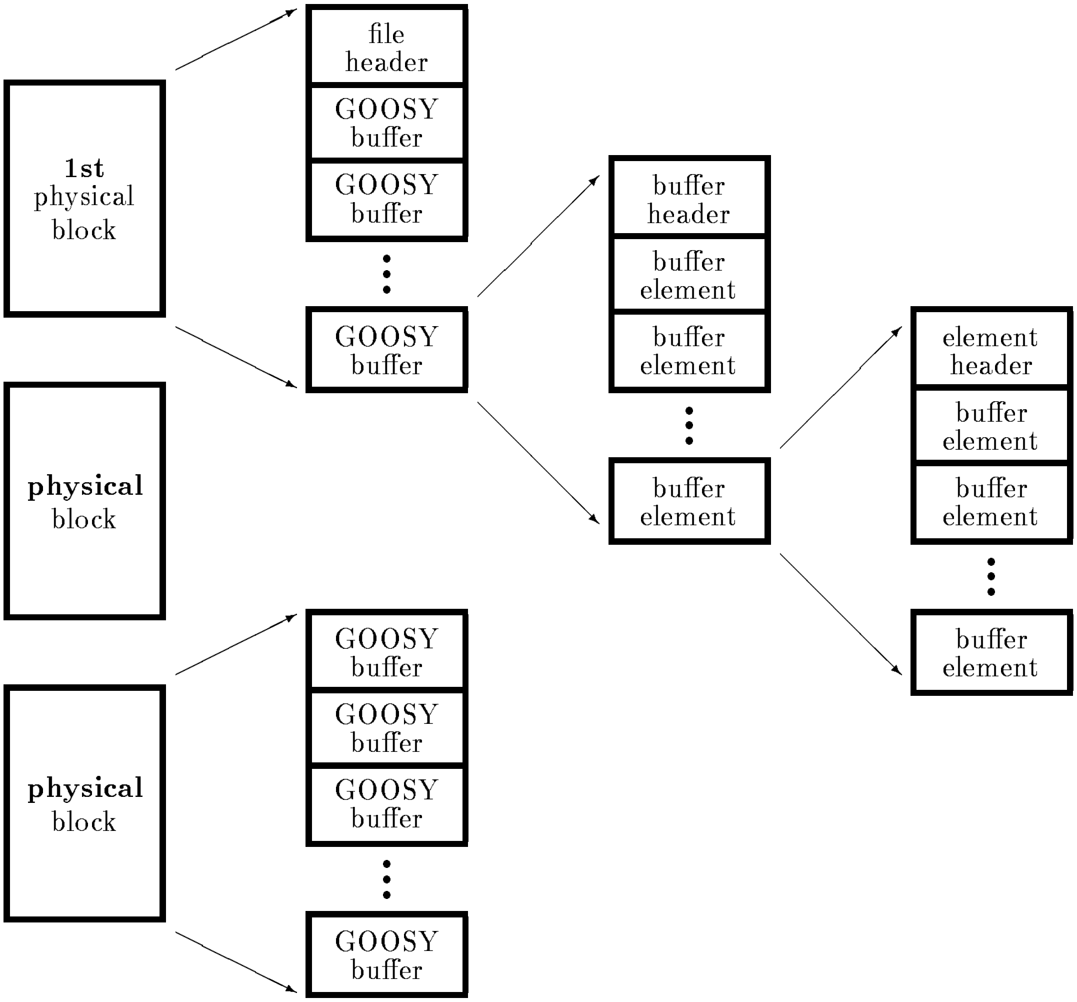


Figure 1.2: The GOOSY data structures of a listmode dump file.

1.3.3 Buffer Header

Buffer Header

31 28 24	20 16	12 8 4 0	Offset
Length of data field (buffer without header)			0
Sub Type		Type	4
Fragment begin	Fragment end	Used Length of Data field	8
Buffer Number for this Type - Sub Type			12
Number of Buffer Elements			16
Current Index			20
Time stamp VMS 64 bit format			24
Time stamp VMS 64 bit format			28
Byte order tag			32
Length of last event			32
2 Longwords reserved			32
2nd Data word		1st Data word	48
. . .			

Figure 1.3: 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.

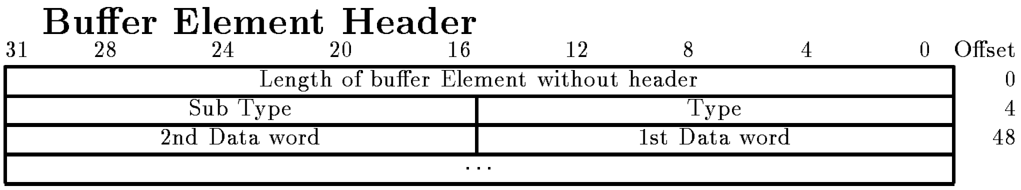
(any).

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             */
/*-------------------------------------------------------------*/
```

1.3.4 Buffer Element Header



31 … 16	15 … 0	Offset
Length of buffer Element without header		0
Sub Type	Type	4
2nd Data word	1st Data word	48
. . .		

Figure 1.4: 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      */
/*-----------------------------------------------------------*/
```

1.3.5 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.

1.4 Buffer Types

1.4.1 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)

1.4.2 File Header Buffer

Figure 1.5 shows the GOOSY File header structure. Note, that the File Header Buffer is a standard GOOSY buffer.

Buffer header information:

Length of data field Depends on buffer length.

(BIN FIXED(31))

Type A number specifying the buffer type. For this file header always = 2000.

(BIN FIXED(15))

Subtype A number specifying the buffer subtype. For this file header always = 1.

(BIN FIXED(15))

Used Length of Data Field Depends on length of comment.

(BIN FIXED(15))

Fragment begin This file header buffer contains *never* incomplete buffer elements. This field is always = 0.

(BIN FIXED(7))

Fragment end This file header buffer contains *never* incomplete buffer elements. This field is always = 0.

(BIN FIXED(7))

Number of Buffer Elements For this file header always = 1.

(BIN FIXED(31))

Buffer Number A current number of buffers of the same type.

(BIN FIXED(31))

Current Index A longword not used.

(BIN FIXED (31)).

Time stamp A quadword for the system time in VAX/VMS binary format. This is the number of 100-nanoseconds since 17–Nov–1858 00:00.

(BIT(64)).

4 Free Longwords ((4) BIN FIXED(31)).

File Header Buffer

31 28 24	20 16	12 8 4 0	Offset
Length of buffer without header			0
Buffer Subtype = 1		Buffer Type = 2000	4
Fragment begin= 0	Fragment end= 0	Used Length of Data field = 1000	8
Buffer Number for this Type - Sub Type			12
Number of Buffer Elements or Fragments of Buffer Elements = 1			16
Not used			20
Time stamp VMS 64 bit format			24
Time stamp VMS 64 bit format			28
4 Longwords reserved			32
...			
Tape label(30 char.)		Used length of tape label	48
Tape label continuation			
...			
File name (86 char.)		Used length of File name	80
File name continuation			
...			
User name (30 char.)		Used length of user name	168
User name continuation			
...			
Date ”dd-mmm-yyyy hh:mm:ss.mm” (24 character)			200
Date continuation			
Run ID (66 char.)		Used length of Run ID	224
Run ID continuation			
...			
Experiment (66 char.)		Used length of Experiment	292
Experiment continuation			
...			
Number of Lines = n			360
Line 1 (78 char.)		Used length of Line 1	364
Line 1 continuation			
Line 2 (78 char.)		Used length of Line 2	
Line 2 continuation			
...			
Line n (78 char.)		Used length of Line n	
Line n continuation			

Figure 1.5: 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          */
/*---------------------------------------------------------------*/
```

1.4.3 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.)

1.4.4 GOOSY Listmode Data Buffers

Listmode data buffers contain buffer elements called *events*. The different event types are described in the next section.

1.5 Event Structures

1.5.1 Event Type 3 (compressed)

Figure 1.6 shows the event structure of type 3. Behind the header there follows one Data Element which is compressed. Two compress modes are supported. One adds a BIT(32) longword for each 32 Longwords. Zero longwords are suppressed and marked in the bitstring. The other adds counter longwords containing the number of following zero or nonzero longwords. **These buffer elements are longword aligned!**

Event Type 3

31 28 24 20 16	12 8 4 0	Offset
Data length		0
Subtype = 1, 2	Type = 3	4
Compression mode		8
Length of uncompressed data		12
First compressed longword		16
...		

Figure 1.6: 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.

1.5.2 Event Type 4

Event Type 4, Subtype 1 (block)

Figure 1.7 shows the event structure of type 4. Behind the header one Data Element follows. The structure is processed as a word array. **These buffer elements are NOT longword aligned!**

Event Type 4, Subtype 1

31 28 24 20 16	12 8 4 0	Offset
Data length		0
Subtype = 1	Type = 4	4
second data word	First data word	8
. . .		

Figure 1.7: Event structure type 4, subtype 1 (block)

Event Type 4, Subtype 2 (no zero's)

Figure 1.8 shows the event structure of type 4. Behind the header one Data Element follows. The structure is processed as a word array. Each data word is specified by an identification number, e.g. an ADC number. **These buffer elements are longword aligned!**

Event Type 4, subtype 2

31 28 24 20 16	12 8 4 0	Offset
Data length		0
Subtype = 2	Type = 4	4
First data word	First data word id	8
Second data word	Second data word id	12
. . .		

Figure 1.8: 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.

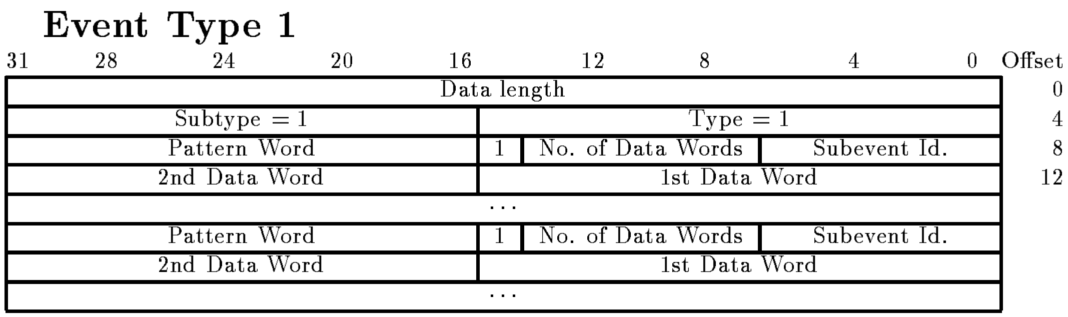
Usage

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.

1.5.3 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 1.9, the buffer element is composed of a header followed by several Data Elements. These Data Elements are produced by the ADC/TDC modules type SILENA 4418x. **These buffer elements are NOT longword aligned!**



Event Type 1

31 28 24 20 16		12 8	4 0	Offset
Data length				0
Subtype = 1	Type = 1			4
Pattern Word	1	No. of Data Words	Subevent Id.	8
2nd Data Word	1st Data Word			12
...				
Pattern Word	1	No. of Data Words	Subevent Id.	
2nd Data Word	1st Data Word			
...				

Figure 1.9: 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.

1.5.4 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 1.10, the buffer element is composed of a header followed by several Data Elements. These Data Elements are produced by the ADC/TDC modules type LRS 4300 (FERA). **These buffer elements are NOT longword aligned!**

Subevent Id A number defining the sub event to which the following subevent belongs. **This byte is NOT longword aligned!**

(BIN FIXED (7)).

# Data Words The number of data words (e.g. ADC values) following the pattern word.

(BIN FIXED (7) but with the highest bit, the event tag bit set to 0 !!).

Event Tag Bit The bit 2^{15} marks the event tag word. This bit is set in subevent header longwords.

(BIT (1)).

Event Type 5

31 28 24 20 16 12 8 4 0

									Offset
Data length									0
Subtype = 1			Type = 5						4
0	1st SA	1st data word	1	# Data Words	0	0	0	Subevent Id	8
0	3rd SA	3rd Data Word	0	2nd SA	2nd Data Word				12
...									
0	1st SA	1st data word	1	# Data Words	0	0	0	Subevent Id	
0	3rd SA	3rd Data Word	0	2nd SA	2nd Data Word				
...									

Figure 1.10: 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.

1.5.5 Event Type 6 (MBD buffer type 6)

Figure 1.11 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))

Event Type 6

31 28 24	20 16	12 8 4 0	Offset
Data length			0
Subtype = 1		Type = 6	4
counter	CAMAC crate	Subevent length	8
Second data word		First data word	12
...			
counter	CAMAC crate	Subevent length	
Second data word		First data word	
...			

Figure 1.11: 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.

1.5.6 Event Type 7 (MBD buffer type 7)

Figure 1.12 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!**

Event Type 7

31 28 24 20	16 12 8 4 0	Offset
Data length		0
Subtype = s	Type = 7	4
Second data word	First data word	8
...		

Figure 1.12: 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.

1.5.7 Event Type 10 (VME)

This structure is composed by the EB. It is mapped by SA$VE10_1 in library GOOINC.

```
/* ================= GSI VME Event header ======================= */
DCL P_SA$ve10_1       POINTER;
DCL 1 SA$ve10_1         BASED(P_SA$ve10_1),
    2 LA$ve10_1_dlen    BIN FIXED(31),
    2 IA$ve10_1_type    BIN FIXED(15),
    2 IA$ve10_1_subtype BIN FIXED(15),
    2 IA$ve10_1_dummy   BIN FIXED(15),
```

Event Type 10, Subtype 1

31	28	24	20	16	12	8	4	0	Offset
Data length [words]									0
Subtype = 1				Type = 10					4
Trigger				Not used					8
Event counter									12
Subevent 1									16
...									
Subevent n									16+x

Figure 1.13: Event Structure

```
    2 IA$ve10_1_trigger BIN FIXED(15),
    2 LA$ve10_1_count   BIN FIXED(31),
    2 IA$ve10_1(LA$ve10_1_dlen-4)    BIN FIXED(15),
    2 LA$ve10_1_next    BIN FIXED(31);
/*-------------------------------------------------------------------*/
```

CAMAC Subevent Structure 10,1

This subevent structure is written by the ROP or the FEP. It is defined in SA$VES10_1 in library GOOINC.

Subevent Type 10, Subtype 1

31	28	24	20	16	12	8	4	0	Offset
Subevent Data length [words]									0
Subevent subtype = 1				Subevent type = 10					4
Control		subcrate		Processor ID					8
CAMAC value				CAMAC module ID					12
...									

Figure 1.14: CAMAC Subevent Structure

```
/* ================= GSI VME Subevent header ======================= */
DCL P_SA$ves10_1       POINTER;
DCL 1 SA$ves10_1         BASED(P_SA$ves10_1),
    2 LA$ves10_1_dlen    BIN FIXED(31),
    2 IA$ves10_1_type    BIN FIXED(15),
    2 IA$ves10_1_subtype BIN FIXED(15),
    2 IA$ves10_1_procid  BIN FIXED(15),
    2 HA$ves10_1_subcrate BIN FIXED(7),
    2 HA$ves10_1_control BIN FIXED(7),
    2 IA$ves10_1(LA$ves10_1_dlen-2)    BIN FIXED(15),
    2 LA$ves10_1_next    BIN FIXED(31);
/*-------------------------------------------------------------------*/
```

FASTBUS Subevent Structure 10,2

This subevents are written from the AEB. The header structure is defined in SA$VES10_1 in library GOOINC.

Subevent Type 10, Subtype 2

31 28 24	20 16	12 8 4 0	Offset
Subevent Data length [words]			0
Subevent subtype = 2		Subevent type = 10	4
Control	subcrate	Processor ID	8
Fastbus module header			16
...			

Figure 1.15: Fastbus Subevent Structure

The following structure maps to the data field. It is defined in SA$vesfb in library GOOINC.

Fastbus module header

31 28 24	20 16	12 8 4 0	Offset
Longwords	Geo.addr.	Module ID	0
1st data word			4
...			

Figure 1.16: 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);
```

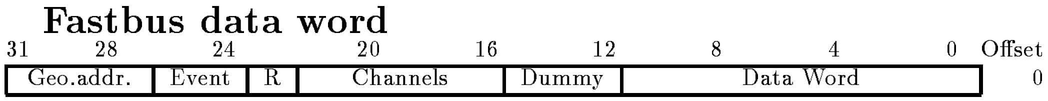


Figure 1.17: Fastbus Data Word

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. */
/*-------------------------------------------------------------------*/
```

1.6 Buffer Element Structures

1.6.1 Buffer Element Type 9000 (Time Stamp)

Figure 1.18 shows the Time Stamp structure (buffer element structure type 9000). **These buffer elements are longword aligned!**

Buffer Element Type 9000, Time Stamp

31 28 24 20 16	12 8 4 0	Offset
Data length=10		0
Subtype = 0	Type = 9000	4
Date ”dd-mmm-yyyy hh:mm:ss.mm” (24 character)		8
Date continuation		

Figure 1.18: 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.

1.6.2 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

1.7 Nonstandard Buffer Structures

1.7.1 Buffer Type 12 (SILENA)

Figure 1.19 shows the subevent structure as produced by one ADC/TDC module of type SILENA 4418x. Several modules produce several subsequent structures. **These buffer elements are NOT longword aligned!**

SILENA Data Structure

31 28 24 20 16		12 8	4 0	Offset
Pattern Word	1	No. of Data Words	Subevent Id.	0
2nd Data Word	1st Data Word			4
. . .				

Figure 1.19: 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.

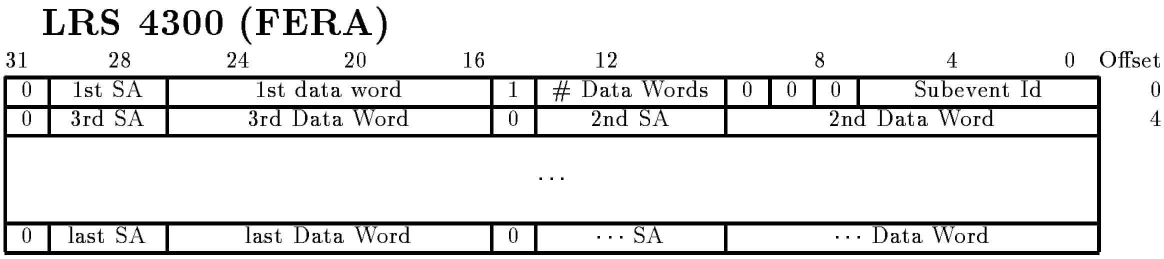
((n) BIN FIXED (15)).

Usage

Not yet used.

1.7.2 Buffer Type 15 (LRS FERA)

Figure 1.20 shows the subevent structure as produced by one ADC/TDC module of type LRS 4300 (FERA). Several modules produce several subsequent structures. **These buffer elements are NOT longword aligned!**



LRS 4300 (FERA)

31	28	24 20	16	12		8		4 0	Offset
0	1st SA	1st data word	1	# Data Words	0	0	0	Subevent Id	0
0	3rd SA	3rd Data Word	0	2nd SA	2nd Data Word				4
				. . .					
0	last SA	last Data Word	0	· · · SA	· · · Data Word				

Figure 1.20: 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.

Index

B

Buffer 4
- element 4, 8
  - calibration 32
  - condition 32
  - data 4, 8
  - Data Element 32
  - header 4, 8, 12
  - picture 32
  - polygon 32
  - spectrum 32
  - time stamp 32
- file header 16
- files 4
- header 10
- non standard 8
- spanning 13
- structure 8
- subtype 10, 12
- type 10, 12, 14

Byte order 5

D

Data
- buffer 20
- Element
  - buffer 20

E

Endian 5

Event 5
- compressed 21
- FERA
  - non standard 34
  - standard 24
- J11 21
- MBD
  - standard 25
  - user defined 28
- SILENA
  - no standard 33
  - standard 23
- spanning 13
- types 14, 21, 23, 24, 25, 28, 33, 34
- VME 28
  - CAMAC 29
  - FASTBUS 30
- word block 21
- word block - no zeros 22

F

FASTBUS
- event 30

File header 16

File names
- IBM 18

fragment 10

I

IBM
- file names 18

M

Message control block 4, 6

S

Spanning 13

V

VME
    event 28
        CAMAC 29
        FASTBUS 30

Contents

1 **GOOSY Data Formats** **3**
1.1 **Introduction** . . . 4
1.1.1 Buffers . . . 4
1.1.2 Buffer Files . . . 4
1.1.3 Message Control Blocks . . . 4
1.1.4 Glossary . . . 4
1.2 **Message Control Block Structure** . . . 6
1.3 **Buffer Structure** . . . 8
1.3.1 Standard Buffers . . . 8
1.3.2 Nonstandard Buffers . . . 8
1.3.3 Buffer Header . . . 10
Structure Declaration . . . 12
1.3.4 Buffer Element Header . . . 12
Structure Declaration . . . 13
1.3.5 Event Spanning . . . 13
1.4 **Buffer Types** . . . 14
1.4.1 Overview . . . 14
1.4.2 File Header Buffer . . . 16
Structure Declaration . . . 20
1.4.3 GOOSY Data Element Buffers . . . 20
1.4.4 GOOSY Listmode Data Buffers . . . 20
1.5 **Event Structures** . . . 21
1.5.1 Event Type 3 (compressed) . . . 21
Usage . . . 21
1.5.2 Event Type 4 . . . 21
Event Type 4, Subtype 1 (block) . . . 21
Event Type 4, Subtype 2 (no zero's) . . . 22
Structure Declaration . . . 22
Usage . . . 23
1.5.3 Event Type 1 (Buffer Type 2, SILENA) . . . 23
Usage . . . 24

1.5.4 Event Type 5 (LRS FERA) 24
Usage 25
1.5.5 Event Type 6 (MBD buffer type 6) 25
Structure Declarations 26
Usage 27
1.5.6 Event Type 7 (MBD buffer type 7) 28
Usage 28
1.5.7 Event Type 10 (VME) 28
CAMAC Subevent Structure 10,1 29
FASTBUS Subevent Structure 10,2 30
1.6 **Buffer Element Structures** 32
1.6.1 Buffer Element Type 9000 (Time Stamp) 32
Usage 32
1.6.2 Buffer with GOOSY Data Elements 32
GOOSY spectrum 32
GOOSY condition 32
GOOSY picture 32
GOOSY polygon 32
GOOSY calibration 32
GOOSY Data Element 32
1.7 **Nonstandard Buffer Structures** 33
1.7.1 Buffer Type 12 (SILENA) 33
Usage 34
1.7.2 Buffer Type 15 (LRS FERA) 34
Usage 35

APPENDIX **35**

Index **36**