GOOSY
Id.: gm_dm
Version: 1.0
Date: September, 16 1987
Revised: June, 28 1988

GSI Online Offline System

GOOSY Data Management

M. Richter

June, 28 1988

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

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 manual is written for advanced GOOSY users and system programmers. It assumes that the reader is familiar with most VAX-VMS concepts and commands. The 'GOOSY Data Management' is a reference manual. It provides all information necessary to implement programs using the internal Data Management structure of GOOSY. All features of GOOSY Data Management are described in detail.

The author would be grateful for any critical comment or any suggestion about this manual.

Chapter 2

Introduction

In a software system for data acquisition and data analysis a large number of differently structured data objects has to be handled. Those data objects could be simple variables like calibration parameters or complex structures like a spectrum. Manipulations like create, delete, modify, copy and show are required for those data objects.

The organization of the data must fullfill the following needs:

- The addressing of Data Elements by names must be possible.
- Several programs must be able to access the data simultaneously (shared memory).
- The organization must be expandable to keep larger amounts of data and to include new data types.
- The data access must be fast, e.g. for the event analysis loop.
- The data organization must keep its integrity by self describing.
- Comfortable tools to maintain a large number of data elements.

The Data Management of GOOSY implemented on VAX is based on a structured Global Section called **Data Base**. The smallest entity which can be located in a Data Base is a **Data Element**, e.g. a single parameter or spectrum data. The Data Elements are stored in sections of the Data Base called **Data Area**. Each Data Area is a cluster of contiguous pages which will be mapped into a program's address space. Data Areas with similar mapping attributes are collected in **Data Pools**, which are contiguous in memory. Information about Data Elements is kept in **Data Element Directories**. The full specification of a Data Element is:

```
node::data-base-name:[DE-directory-name]data-element-name(index).member
```

(see appendix A on page 39).

2.0.1 Glossary

Data Base A formatted VMS global section. The Data Base name is a logical name of a VMS global section or the global section name itself.

Global Section Part of memory which can be shared by several processes. Provided by VMS.

Global Section File Each global section is created as a file. Parts of the section can be mapped into a process virtual address space. Global section pages are paged to the global section file.

Data Base Area Contiguous number of pages in the Data Base (global section file).

Data Base Pool Composed of several areas. Smallest entity which can be mapped by a process. One pool has for one process one access mode (Write or read only). If a pool runs out of space, one more area is chained to that pool.

Data Element Piece of data in the data base. It has a name which is kept with other information in a directory. The data part is kept in a pool (area). The structure of a data element is described by a PL/1 structure declaration.

Data Element Member Member of a data element structure.

Data Element Array Data Elements can be indexed in up to two dimensions. Each element of such an array has its own slot in the directory. Therefore the data structures could be different.

Data Element Type A data element describing a PL/1 structure. It is created from a PL/1 structure declaration. Data Elements are created with that data structure by refering to the corresponding Type.

Data Element Directory Information about Data Elements, Areas, Pools, Data Types and Directories is kept in directories.

Mount/Dismount Mount a Data Base means to create a global section. The global section file (data base) must exist.

Attach Attach a data base, pool, directory, Data Element means to map the appropriate parts of the data base into the memory of the attaching process. The resulting pointers are kept in a local mapping context structure normally invisible to the user. Only Data Element pointers are returned.

Locate Locate a Data Base, pool, directory or data element means to get the pointer to the object and/or get an identification number. This number is valid and unique during the lifetime of the object. It can be used for a fast Locate to get the pointer.

Chapter 3

Data Management Functionality

The Data Base Management is a collection of software components, which allow to manipulate data objects in a multiprocess/multiprocessor environment.

The basic functionality is:

- Create and delete data objects at various complexity levels:
  - **Data Elements**, which are simple variables or PL/I like structures,
  - **Data Areas**, which are contiguous clusters of Data Elements, the smallest mapping units of Data Bases,
  - **Data Pools**, which are collections of Data Areas with the same mapping protection,
  - **Data Element Directories**, which are collections of similar Data Elements,
  - **Data Bases**, which are collections of Data Areas with Directories,
  - and finaly the **Data Environment**, which is the collection of all active Data Bases and all correlated processes of an user.
- Locate an already created Data Element.
  - either by name with a hierachically structured **Directory** system, which is part of every Data Base
  - or by indices of the Master Directory and the corresponding Data Element Directory.
- Access a created or located Data Element.
  - directly via inline code in PL/I, which has been generated by a preprocessor. The direct access mechanism must be as efficient as possible, because it is used for very often accessed Data Elements (e.g. in an event analysis loop). For each Data Area of a Data Base a **local Mapping Context** must be kept in the accessing program containing pointers to the start of mapped Data Areas (see appendix C on page 47).

  - indirectly via procedure calls, which read from or write into Data Elements.
  - by several commands for general Data Elements and for specific Data Elements.

- Ensure Data Base integrity.

  A locking mechanism synchronizes operations in a multiprocess system (see appendix B on page 43). This is necessary due to several reasons:

  - System integrity
    The integrity of a Data Base structure must be ensured under all circumstances. Therefore the access to system structures has to be done by system procedures which will do the locking.
  - Data integrity
    The integrity of the application data has to be ensured as far as possible.
  - Data copy validity
    If a Data Element is copied into a local PL/I structure, than modified, and finaly restored, it has to be ensured that no race conditions will occur.
  - Local pointer validity
    For an efficient direct access of Data Elements it is necessary to map PL/I structures to the Data Elements by pointers which locate them in the process' virtual address space, the so called **Mapping Context**. The validity of these pointers must be ensured for all modifications of a Data Element (see appendix A on page 39).

- Provide Data Protection.

  A write protection mechanism allows to protect data objects. There are two protection object classes:

  - Structure protection
    As mentioned above the structural integrity of a Data Base must be ensured under all circumstances. Therefore the system components like the Area Directory, the Pool Directory, and the Data Element Directories and with them the Data Element Descriptors and the Type Descriptors are collected in seperate system Data Areas of the Directory Pool. This allows to use hardware protection mechanisms to prevent unintended modifications.
  - Contents protection
    As discussed for system Data Areas, any other Data Area may be write protected in certain processes.
    E.g. Data Areas of the spectra Data Pools would be write accessible for the analysis process but write protected for a display process. A Data Area containing parameters may be write protected for an analysis process but write accessible for an utility process to change those parameters. Furthermore the protection of a Data Area may be changed during the lifetime of a process, for example write accessible during the setup phase but write protected afterwards.

- Command Interface.

  The Data Management functions are all accessible by user written programs or by commands. The command interface is called the **Data Manager**. The Data Manager can be called from DCL level by the command:

```
$ MDBM
```

  which prompts with

```
SUC: DBM>
```

  or within a GOOSY environment with the component MGOODBM. With the `Next Screen` key of a VT220 compatible terminal keyboard you enter a command menu. A detailed description about the Data Base Manager can be found in chapter 7 on page 25. See also the manual 'GOOSY Introduction'.

Chapter 4

Data Management Implementation

The Data Management functions are available at three different layers:

1. Commands
   These provide a simple interactive access to all functions by commands.
2. Procedures
   Basic operations implemented in procedures provide to extend access funtions.
3. Macros
   Data Elements may be accessed in application codes directly. The name of the Data Element is referenced in macros or marked by a prefix. The precompiler generates PL/I structures which are mapped over the Data Element.

The implementation details are obviously system dependant:

- VAX/VMS:
  The Data Bases are implemented with global sections, the protection via page protection (see appendix A on page 39), and the locks via the VMS lock manager (see appendix B on page 43).
- IMPORTANT NOTE:
  All information which is needed to access the Data Elements of a Data Base is stored in the same Data Base. This selfconsistency allows to implement a simple and general procedure which converts between various data representations.

4.1 Commands

4.2 Procedures

4.3 Macros

Chapter 5

Data Management Organization

The Data Management is organized internally in a hierachical manner. Each component of the organization can be addressed by name. The names are collected in named Directories, the Directory names in a master directory. The data regions are split in Data Areas, the Areas are bundled in Data Pools. The names of the Data Areas and the Data Pools are collected in the Area Directory and the Pool Directory, respectively. A Home Block keeps the entry information about the main Directories and the storage information for the Data Areas, i.e. the Data Base usage. Figure 5.1 on page 16 gives a simple overview about the Data Managment Organization.

In the following, the components of the Data Management are described briefly. For a detailed description see chapter 6 on page 21. The corresponding PL/I structures are listed in appendix C on page 47.

- **Member Value** (MV)

  This is the smallest entity, which can be **located** and **accessed** via the Data Management. A Member Value is a simple variable of a specific Data Type. The supported Data Types are:

  - BIN FIXED(7)
  - BIN FIXED(15)
  - BIN FIXED(31)
  - BIN FLOAT(24)
  - BIN FLOAT(53)
  - BIT(*) ALIGNED
  - CHARACTER(*)
  - CHARACTER(*) VAR
  - OFFSET
  - UNKNOWN

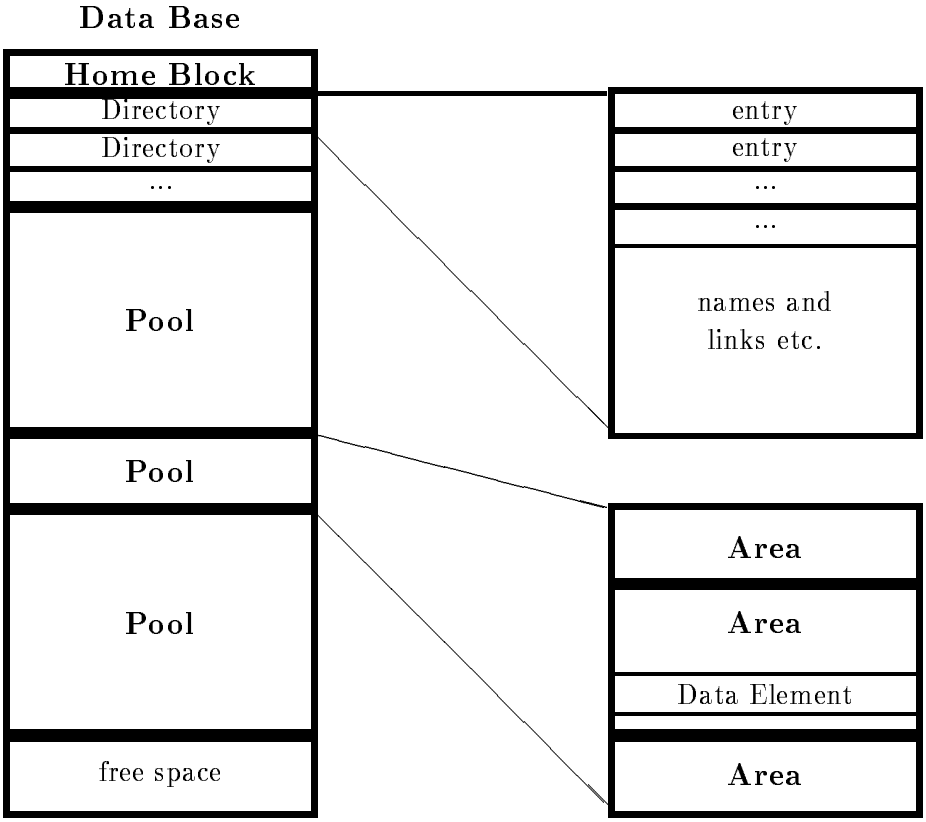

Figure 5.1: The simplified structure of a GOOSY Data Base.

- **Element Member** (EM)

  An Element Member is a Member Value or a 1 to 8 dimensional array of Member Values.

- **Data Element** (DE)

  A Data Element is the basic entity which can be manipulated by the Data Management. The structure of a Data Element is defined by a Data Type.

  The three basic Data Element forms are:

  – simple Data Element
    Containing only one Member Value or Element Member and corresponding to a simple variable or a simple array of PL/I.

- complex Data Element
  Containing several Element Members and corresponding to a structure of PL/I.
- indexed Data Element (name arrays)
  This is an array of Data Elements corresponding to a pointer array of PL/I referencing structures of the same Data Type.

The following list shows some examples of Data Elements in GOOSY.

- Coordinates and parameters:
  They are implemented with simple Data Elements.
- Spectra
  A spectrum consists of a header in a protected pool and a data array.
- Conditions
- Command definitions
- Display pictures
- Buffer pools
- Control structures

- **Data Area** (DA)

  A Data Area is the smallest entity which can be made accessible (mapped) to a program.

  All information which requires a specific combination of **protection** attributes is normaly collected in one Data Area. A Data Area is a storage area in which Data Elements can be **created**.

  Every Data Area consists of an Area header and a free allocation part. The header keeps the information about the following items:

  - Length of the Data Area.
  - Index number of the Data Area in the Area Directory.
  - Version number of the Data Area (for integrety check).
  - Relative address of the start of the free allocation part.
  - Informations about the allocation within the Data Area.
  - The value of the byte cluster for the Data Area. A byte cluster is the smallest number of bytes which can be allocated in the Data Area. The byte cluster value depends on the Data Elements to be allocated in the Data Area. For single parameters it might be 4 bytes, for spectra it might be 512 bytes.
  - An allocation bit map with one bit per byte cluster.

  (see appendix C section C.3 on page 58).

- **Data Pool** (DP)

  A Data Pool is the collection of Data Area which requires the same specific combination of **protection** attributes. All Data Areas of a Data Pool are logical linked together. If a Data Area is short of space, a new Data Area can be created with the same attributes than the previous one. This new Data Area will be linked to the same Data Pool. A user normally knows the Data Pool only, not the Data Area.

- **Data Base** (DB)

  All information associated with one application is normaly collected in one Data Base. A Data Base is a storage area, in which Data Areas can be created.
  Every Data Base has the following **protected system Data Areas**:

  Home Block The Home Block is a specific part of the Data Base. It is not a normal Data Area. The Home Block is always located at the begin of every Data Base and contains all information to **locate** other Data Areas, especially the **Directories**. It also keeps an allocation bit map of the whole Data Base and general informations like the section file name, creation date and time etc. (see appendix C section C.2 on page 56).

  Area Directory This Directory contains the relative addresses, the length in pages (512 bytes), the allocation cluster size, and the names of all Data Area for the whole Data Base. It also keeps the Data Pool link information for each Data Area (see appendix C section C.5 on page 64).

  Pool Directory This Directory contains the names of all Data Pools for the whole Data Base and the minimum size in bytes of any Data Area in this Data Pool. It also keeps the link information for the first Data Area of each Data Pool (see appendix C section C.6 on page 68).

  Master Directory This Directory contains the names of all Data Element Directories and the relative addresses of their Data Element Directories for the whole Data Base (see appendix C section C.7 on page 72).

  Data Element Directories Each of these Directories contain the names of all Data Elements, the relative addresses of their Data Areas, the relative addresses within the Data Areas, and other Data Element information for all Data Elements of one Data Element Directory. This information is called **Data Element Descriptor** or **Directory Entry**.

  There are three additional extensions possible for each entry in a Data Element Directory:

  – Any extension of the Data Element Descriptor. These extensions may be of any length. They are characterized by an extension type. All extensions of a Data Element Descriptor are linked. The character string of the name of a Data Element is located within such an extension.

- A queue of Data Elements of the same Data Element Directory, i.e. in the same Data Element Directory. This allows to bind unnamed Data Elements to named Data Elements, e.g. a named spectrum header and its unnamed spectrum limit definitions.
- A link to Data Elements of any Data Element Directory. This allows logical correlations of Data Elements, e.g. conditions linked to a spectrum.

(see appendix C section C.8 on page 75).

Type Directory The Type Directory is a specific Data Element Directory which contains the Data Types of all Data Elements in the Data Base. Each new Data Type must be inserted in the Type Directory before it can be used by a Data Element (see appendix C section C.9 on page 81).

- **Data Environment** (DEN)

  The Data Environment is the collection of all Data Bases and all processes which are visible for a user.

Chapter 6

Detailed Description

6.1 Home Block

6.2 Area Directory

6.3 Pool Directory

6.4 Master Directory

6.5 Data Element Directory

6.6 Type Directory

6.7 Type Descriptor

(See appendix C section C.10 on page 85)

- Type Descriptor length
- Data Type name
- number of members
- for every member:
  - member name
  - Data Type of member
  - dimensionality of member
  - for every dimension or size: bound or refer object

6.8 Data Element Descriptor

- version number of Data Element
- offset to the Data Element Descriptor extent
- index of the Data Area in which the Data Element is allocated
- offset within this Data Area
- size of Data Element
- Data Type of Data Element
- status flags:
  - named or queued Data Element
  - member of a name array
  - protected against deletion
- link information
- queue forward and backward reference
- name for named Data Elements

6.9 Data Base Usage

The following list summarizes the Data Base usage in GOOSY:

- **Environment Data Base** (not yet implemented)
  One Data Base per GROUP, which contains the **Data Base Directory** and all other information, which must be handled on group level.
- **Command Data Base** (not yet implemented)
  One Data Base per ENVIRONMENT, which contains all command descriptions.
- **Profile Data Base** (not yet implemented)
  One Data Base per USER, which contains all user specific information:
  - User specific command attributes like replaceable defaults (parameter specific or global) or the process in which a command is executed.
  - Logical name table.

- **User Data Base(s)**
  Those Data Bases may contain Data Elements, which are accessed in more than one acquisition process, e.g. global parameters.

- **Dump Data Base(s)**
  Those Data Bases are alike a Local Data Base, but are used to store Data Elements for a longer time. The structure is equal, but for some Data Elements a different representation may be used (e.g. compressed spectra).

Chapter 7

Data Base Manager

7.1 Data Base Manager Introduction

There are two interfaces to the data base management:

1. Command Interface: Data Base Manager.
2. Program Interface: Data Management Routines.

The Data Base Manager is a program which can be called from DCL level as a stand alone program or as a component of a GOOSY environment. It executes all commands provided by GOOSY to handle data bases.

7.2 Data Management

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 created for the Data Base name. To activate a Data Base it must be **mounted**. It is dismounted during a system shutdown or by command. If a Data Base runs out of space, it can presently NOT be expanded.

The data region of a Data Base is splitted into **Pools**. All Data Elements are stored in Pools. A Pool (and all Data Elements in the Pool) can be accessed by a program with READ ONLY protection or with READ/WRITE protection. Pools must be created. They are automatically expanded if necessary, up to the space available in a Data Base. Similar to a VMS disk, Data Elements of a GOOSY Data Base are organized in **Directories**. The user must create Directories to use them. A diagram of the simplified Data Base structure is shown in figure 5.1 on page 16.

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

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

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

```
$ CREDB basename filename size[KB]
       /SPECTRUM=s     ! maximum number of spectra
       /PICTURE=p      ! maximum number of picture frames
       /CONDITIONS=c   ! maximum number of conditions
```

This command creates the Directories $SPECTRUM, $CONDITION, $PICTURE, $DYNAMIC and the Pools $SPEC_POOL, $COND_POOL, $PIC_POOL and $DYNAMIC. For additional information for this command type `HELP CREDB`. An example is shown in section **??** on page **??**. Data Bases are accessible on one node by all programs started by the same VMS user on that node. Before accessing a Data Base, it must be mounted (already done by `CREDB`). This is done by issuing the DCL command:

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

GOOSY should then tell you that the Data Base `basename` has been mounted as a global section and is opened and ready. `filename` is the Data Base file name (a VAX/VMS global section file). It is strongly recommended that the Data Base name is the same as the file name and that the

filename has the type .SEC. The Data Base remains mounted until the node is rebooted or the Data Base is dismounted explicitely by the DCL command:

```
$ MDBM DISMOUNT BASE basename
```

NOTE, however, that the Data Base remains mounted as long as there are programs active using it, even if the message says that there is 'no such (global) section'. Only processes running on the same node of a cluster are allowed to share Data Bases. If you need a Data Base already mounted on a different node, change either to that node or dismount the Data Base on the other node and mount it on your node.

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

7.3 GOOSY Data Elements

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

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

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

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

7.3.1 Conditions

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

Window Conditions

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

Multiwindow Conditions

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

Pattern Conditions

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

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

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

Function Conditions

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

Polygon Conditions

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

Composed Conditions

This may be any boolean expression of other conditions.

7.3.2 Polygons

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

7.3.3 Spectra

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

Spectra are created as digital or analog spectra.

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

7.3.4 Calibrations

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

7.3.5 Pictures

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

7.3.6 User Defined Data Elements

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

1. Put the PL/1 source declaration of the Data Element in a text library. The name of the structure should be used as the name for the library module. The declaration must declare a based structure. A base pointer may be specified (should be STATIC).
2. Create a Directory in Data Base (optional)
3. Create a Pool in Data Base (optional)
4. Create the Data Element Type, using the new PL/1 structure in the text library.

5. Create the Data Element of the new Type.

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

7.4 Data Base Manager

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

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

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

```
Subcommands available ================================================
$ *          :  * :ATTACH,CALL,DCL,DEBUG,DEFINE,DIRECTORY,EXECUTE,EXIT,..
CALCULATE * :  * :SPECTRUM
CALIBRATE * :  * :SPECTRUM
CLEAR *      :  * :CONDITION,SPECTRUM
COPY *       :  * :ELEMENT,MEMBER,SPECTRUM
CREATE *     :  * :AREA,CONDITION,BASE,DIRECTORY,DYNAMIC,ELEMENT,...
DELETE *     :  * :CONDITION,DYNAMIC,ELEMENT,GLOBAL_SECTION,LINK,POOL,..
DECALIBRATE :  * :SPECTRUM
DISMOUNT *  :  * :BASE
DUMP *       :  * :SPECTRUM
FREEZE *     :  * :CONDITION,SPECTRUM
HELP         : Access VMS HELP facility
LOCATE *     :  * :BASE,DIRECTORY,ELEMENT,ID,POOL,QUEUEELEMENT,TYPE
MENU         : Enter menu
MODIFY *     :  * :DIRECTORY
MOUNT *      :  * :BASE
SET *        :  * :CONDITION,LETTERING,MEMBER
SHOW *       :  * :AREA,CONDITION,DIRECTORY,DYNAMIC,ELEMENT,HOME_BLOCK,.
UNFREEZE *  :  * :CONDITION,SPECTRUM
UPDATE *     :  * :BASE,DYNAMIC
************** End of list ******************** End of list ***
Command:
 PF2: Help, PF3: Enter command, PF4: Break, ENTER: Previous menu
Subcommand :
```

You should play around in the menu. If you execute a command, the full command line will be displayed on top of the screen.

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

7.4.1 CREATE Commands

Create Directories

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

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

Create Pools

All Data Elements are allocated in Pools. Normally the default Pools created by command `CREDB` are adequate. One may, however, create additional private Pools. The poolsize is no limit of the Pool, because it is extended automatically. One should at least specify the size of the largest Data Element to be allocated in the Pool.

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

Create Data Element Types

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

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

Create Data Elements

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

Create Conditions

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

```
CRE COND WINDOW c (1,1000) 1          ! 1 subwindow
CRE COND WINDOW c (1,1000) 2          ! 2 subwindows
                                      ! both (1,1000)
CRE COND WINDOW c(10) (1,100)         ! 10 cond., 1 subw.
CRE COND WINDOW c (1,100,1,200)       ! 2 subwindows
CRE COND MULTI  c (1,1000) 100        ! 100 subwindows
CRE COND PATTERN c '1'B               ! 1 subpattern
                                      ! padded right with 0
```

```
CRE COND PATTERN c '1'B 2              ! 2 subpatterns
CRE COND PATTERN c '1'B INV='1'B /ANY ! invert first bit
CRE COND PATTERN c '11111'B /IDENT     ! identical match
CRE COND COMP c "a | (x & y)"          ! a, x, y must exist
```

Create Spectra

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

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

Create Dynamic Lists

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

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

Create Dynamic Entries

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

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

This declaration is in our library TPRIV in module SX$EVT. We refer in the following examples to Data Element EVT of the above Type. We assume that conditions c,w,a(1:10) and spectra s and s2 already exist.

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

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

```
                PARAMETER=([eva]evt.naj(1),[eva]evt.geli(1)) -
                CONDITION=c
                                        ! 2-dim. spectrum
```

Create Pictures

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

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

Create Calibrations

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

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

7.4.2 SHOW Commands

```
SHOW CONDITION *                      ! list of all conditions
SHOW CONDITION c /FULL                ! full information
SHOW CONDITION * /WINDOW              ! window cond. only
SHOW SPECTRUM *                       ! list of all spectra
SHOW SPECTRUM * /FULL                 ! full information
SHOW SPECTRUM s /DATA                 ! spectrum data
SHOW DYNAMIC LIST list *              ! all entries
SHOW DYNAMIC LIST list SPECTRUM       ! all spectrum entries
SHOW PICTURE *                        ! list of all pictures
SHOW PICTURE pict /FULL               ! full information
```

```
SHOW ELEMENT [EVA]*                    ! list of elements in [EVA]
SHOW ELEMENT [EVA]EVT /DAT             ! contents of [EVA]EVT
SHOW TABLE                             ! Show all counters of
                                       ! spectra and conditions
SHOW MEMBER [EVA]EVT.GELI(1)           ! show contents
```

7.4.3 CLEAR Commands

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

7.4.4 DELETE commands

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

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

7.4.5 Miscellaneous Commands

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

Appendix A

Mapping Concept

A Data Base is implemented on a VAX under the VMS operating system as a Global Section. A Global Section consists of a Section File containing all data which should be shared by many processes. This file gets a group or a system wide Section attribute by an initiating process which calls the VMS system service procedure SYS$CRMPSC. This mounting of a Data Base will be done by the GOOSY procedure M$MODB. Each process may map parts of a Data Base (pages of a Global Section) which it wants to share with other processes. The start block within the Data Base and the length in blocks (512 byte page) to be mapped must be known. This information is held in the Data Base Directories. The mapping can be done also by the VMS system service procedure SYS$CRMPSC. This is done by the GOOSY procedure M$MPDB, which is used by all attach modules M$ATxx. The procedures return the address of the first and the last virtual memory location to which the part of the Global Section was mapped. This array of two pointers is held in the local mapping context structure SM$DBMC of each process (see appendix C section C.1 on page 48).

To map a part of a Data Base, the access mode to this part must be defined also. There are two possible access modes:

1. read only access
   This mode allows to read all data of the mapped part of the Data Base. It prohibits any change of the data by the mapped process. This mode is useful e.g. for SHOW command procedures.

2. read/write access
   This mode allows to read and write all data of the mapped part of the Data Base. It allows any change of the data by the mapped process. This mode is necessary e.g. for CREATE command procedures.

To set one of these access modes, one has to map the part of the Global Section. If this part of a Data Base was mapped already with another mode, one has to unmap it first (detach) and then map it again (attach) with the new mode. Thereby the start and end virtual address will be changed. All correlated local mapping contexts will be changed after these operations. Therefore,

Data Element Directories and Data Elements of this part of the Data Base must be located again after a remap.

In general the following steps should be done to get the correct mapping context:

1. Attach a Data Base (M$ATDB):

   The Home Block and the main Directories (Area-, Pool-, Master-, Type-Directory) will be mapped with the required access mode. To create a Directory, a Data Element, a Data Element Type, or a Pool, the main Directories and the Home Block must be mapped with write access.

2. Get the Data Element Directory indices and the Pool indices of all Data Elements you want to deal with (M$DEID):

   Each Data Element has an entry in a Data Element Directory. Each Data Element with non-zero data has its data in Pools.

3. Attach all Data Element Directories you need (M$ATDI):

   A Data Element Directory must be mapped at least with read access to use any Data Element index operation. To create Data Elements their Directories and the main Directories must be mapped with write access.

4. Attach all Pools of all Data Elements you need (M$ATPO):

   Data Element data are located in Pools. A Pool is a bunch of Data Base Areas which are mapped identically, i.e. the data of all Data Elements of a Pool are mapped with the same access mode.

5. Locate all Data Elements you need (M$LOID or M$LODE):

   The locate procedures return the virtual address of a Data Element in its Pool. The Pool must have been attached separately in advance.

If a Directory or a Pool has to be remapped with a different mapping access one needs the following steps:

1. Detach the Directory or Pool (M$DADI, M$DAPO):

   The local mapping context of a Directory or Pool is invalid after a detach.

2. Attach the Directory or Pool with the right access mode, M$ATDB, M$ATPO
   The local mapping context of a Directory or Pool is renewed.

3. Locate all involved Data Elements again (M$LOID):

   If a Pool was remapped, all Data Elements having data in this Pool must be located to get the correct virtual data pointers again.

If the Home Block and all main Directories have to be remapped with a different mapping access one needs the following steps:

1. Detach the whole Data Base (M$DADB):
   All mapping information is lost after the detach of a Data Base.
2. Attach the Data Base again with the new access mode (M$ATDB).
3. Attach all Data Element Directories and the Pools you need (M$ATDI, M$ATPO).
4. Locate all Data Elements you need (M$LOID).

Appendix B

Locking Concept

Since a Data Base is implemented under VAX-VMS operating system by a Global Section, many processes have access to the data stored in the Data Base. Each process having write access may modify the data. It can also modify the internal organization of a Data Base by adding or deleting Data Elements and/or Data Element Directories. These possibilities require a protection mechanism between different processes to keep the integrity of Data Bases during access. Under VAX-VMS operating system the lock manager is used to protect processes during access of parts of the Data Base.

The VMS lock management system services allow different processes to synchronize their access to shared resources. To synchronize access to resources, the lock management services provide a queuing mechanism allowing processes to wait in a queue until a particular resource is available, i.e. another process has finished its access to the resource.

In case of a Data Base a resource is one of the following parts:

- the whole Data Base (name: LCK_'db-name')
- the Home Block (name: $LCK_HBV1)
- the main Directories, Area-, Pool-, Master-Directory (name: $LCK_MAIN_DIR)
- each Data Area (name: $LCK_AREA_'area-index')
- each Data Element Directory (name: $LCK_EDIR_'dir-index')
- each Data Element (name: $LCK_DE_'de-index')

The Enqueue Lock Request system service (SYS$ENQ) is used to make lock requests and the Dequeue Lock Request system service (SYS$DEQ) is used to cancel lock requests. A resource lock gets a name by the process requesting the lock.

Each lock is system wide identified by a unique Lock Identification number. This number is created by the VMS operating system during the creation of a lock. The Lock Identification numbers of all Data Base locks created by a process are stored in the local mapping context structures SM$DBMC (see appendix C section C.1 on page 48).

A requested lock has an associated lock mode. The lock mode indicates how the process wants to share the resource with other processes. This lock mode can be requested and converted by the SYS$ENQ system service. The lock management system services compare the lock mode of a newly requested lock or converted lock to the lock modes of other locks from other processes with the same resource name.

- If no other process has a lock on the resource, the new lock is granted.
- If other processes have a lock on the same resource and the mode of the new request is compatible with the existing locks, the new lock is granted.
- If other processes already have a lock on the resource and the mode is not compatible, the new request is placed in a queue waiting for the resource to become available. Depending on the system service the requesting process can wait until the resource is available (SYS$ENQW) or it can continue (SYS$ENQ) until it is notified (Event Flag and/or AST) that it can access the resource.

The Data Base resources are organized in a hierarchical order. This is reflected in the lock's parentage.

1. the whole Data Base lock is the parent of the Home Block lock and the main Directories lock,
2. the main Directories lock is the parent of all Data Element Directory locks and all Area locks,
3. each Data Element Directory lock is the parent of all locks of the Data Elements of this Directory.

A lock request has one of the following associated lock modes:

1. Null Mode
   This mode grants no access to the resource. The Null mode is typically used as an indicator of interest in the resource, or as a placeholder for future lock conversion.

   During the attach to a Data Base the locks for the whole Data Base, the Home Block, the main Directories, all Data Element Directories, and all Areas are created with Null mode. The locks for the Data Element are created with Null mode during the attach to a Data Element Directory.

2. Concurrent read
   This mode allows sharing read of the resource with other readers. Writers are allowed access to the resource (unprotected read).

   Not yet used in GOOSY.

3. Concurrent write
   This mode allows sharing write to the resource with other writers. Writers are allowed access to the resource (unprotected write).

   Not yet used in GOOSY.

4. Protected Read
   This mode allows sharing read of the resource with other readers. No writers are allowed access to the resource.

   If a Directory search operation or any Directory index is in use, the lock of this Directory is set to Protected Read. After the locate of a Directory M$LODI the Directory lock is in Protected Read on return. After a locate of a Data Element of this Directory M$LODE, M$LOID, M$LODA, M$LOQE, M$LOQA, the lock of the Directory is set to Null mode and the lock of the Data Element is set to Protected Read on return. If the Data Element is no longer in use, its lock must be explicitly converted to Null mode. Otherwise no change can be done on this Data Element by other processes.

5. Protected Write
   This mode allows sharing write to the resource with other concurrent read mode readers. No other writers are allowed access to the resource.

   Not yet used in GOOSY.

6. Exclusive
   This mode allows write to the resource. No other writers or readers are allowed access to the resource.

   The Home Block lock is set to Exclusive if any new Area will be created or deleted. The Home Block lock is always Null mode on return of all procedures setting the lock to Exclusive.

   The Main Directories lock is set to Exclusive if any new Area, a new Pool or a new Data Element Directory will be created or deleted. The Main Directories lock is always Null mode on return of all procedures setting the lock to Exclusive.

   A Data Element Directory lock is set to Exclusive if any new Data Element will be created or deleted. A Data Element Directory lock is always Null mode on return of all procedures setting the lock to Exclusive.

   An Area lock is set to Exclusive if any new Area, any Directory or any Data Element will be created or deleted. An Area lock is always Null mode on return of all procedures setting the lock to Exclusive.

To show current locks use the DCL commands:

```
$ CWHAT LOCKS
```

or better

```
$ MLOCKS
SUC: LOCKS>                    ! Now enter '$ MENU' to get a command menu
```

Appendix C

PL/I Structures

A Data Base is a collection of addressable data in a data file. On the VAX, this file is defined as a Global Section. A part of a Data Base or the whole Data Base may be mapped to a process' data space with different access modi (read only, read/write). The main mapping unit of each Data Base is a page, i.e. 512-bytes blocks of data. These pages are equal to DEC's file blocking size. A process can map only multiples of pages.

The first pages of each Data Base are reserved for general Data Base information, the Home Block.

To allocate variable blocks of data in a Data Base, a Data Base is divided into Areas. An Area consists of one or more pages. An Area has a header information, which defines the Area by its length, an identification version, and an Area internal data clustering. A data cluster is the unit of data bytes (any fixed number of bytes) which will be allocated or freed within the Area. For each data cluster in an Area, a bit is reserved in a Bit Map. The Bit Map is located in the Area header. If n data clusters are allocated in an Area, the n corresponding bits of the Bit Map of this Area are set to '1'B. If n data clusters are freed again, the n corresponding bits of the Bit Map of this Area are set to '0'B. By this mechanism the space of an Area can be controlled easily, freed data clusters can be reused.

The same Bit Map mechanism is used to control the overall Data Base allocation. The data cluster is one page (512 bytes) is this case. The Data Base Bit Map is part of the Home Block information.

To address data within a Data Base, naming paths are included. Each Area of a Data Base has a name. These names are ordered in an Area Directory together with mapping and Area cluster information.

Several Areas are combined to Area clusters, the Pools of a Data Base. A Pool is a logical cluster of Areas with the same mapping attributes. The names of the Pools, the identification indices of the first Area of each Pool, and the minimum allocation size of an Area within the Pool are kept in the Pool Directory. Pools are the only mapping entities normally visible to an user.

A Data Element, the smallest by-name addressable data entity in a Data Base, is defined by a Data Element Descriptor. This descriptor is part of a Data Element Directory. The identification

version number, a possible name, attribute flags, the mapping information (Area and offset within this Area for a Data Element), the Type Descriptor identification index, informations about possible links to other Data Elements, information about name arrays of Data Elements, and the identification indeces of queued, not named Data Elements, are defined in the descriptor.

Data Elements are logically grouped into Directories. A Directory is a collection of Data Elements. The Master Directory contains the names of all Directories in a Data Base.

The name of a Data Element is defined by:

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

a queued Data Element is defined by the name of the Data Element building the head of the queue, the Data Type of the queued Data Element, its name and its index

```
queue-head->type:name(index).member
```

Each Directory of a Data Base has a fixed header and n fixed blocks for each entry. The entry is addressed by an identification index of the items listed in the Directory. This index longword can be used as an index to the Directory to get fast access to the entries. The identification version is a longword which will be incremented each time the item is changed. By this, the validity of an item can be checked (e.g. the mapping context).

In the following all lengths are given in number of bytes unless otherwise defined.

C.1 Data Base Mapping Context

Up to L_SM$ALL_DBMC_MAX_NUMBER different Data Bases might be attached by one process and the same number of different dynamic lists might be defined for this process. This fixed number defines a structure array of two pointers, one array member for the base pointer of the Data Base mapping context and one for the base pointer of the list structure. This structure array of base pointers is based on the fixed, external pointer PM$DBMC. Both structures, the Data Base mapping context and the dynamic list, are independent of each other. Their base pointers are just within the same pointer array to have only one external pointer for all other pointer values. When a Data Base or a dynamic list is attached to a process, the mapping context structures are allocated and the corresponding pointers and REFER elements are initialized. Using a Data Base requires a copy of the pointer of its structure array member P_SM$ALL_DBMC_MAIN(DB-index) into the fixed pointer P_SM$DBMC_MAIN. (For the dynamic list the pointer P_SM$ALL_DYNAMIC(list-index) will be used.) This pointer is used as the base pointer to the structure of the mapping contexts for the Home Block and the main Directories. The copy of the pointers from mapping array must be done by each module using the Data Base or dynamic list mapping context structures. The loading of the mapping array and the allocation of the mapping structures is done by the Data Base attachment procedure. The pointer PM$DBMC is an EXTERNAL variable, available in each module of a process. Therefore PM$DBMC is also STATIC and will be set to NULL() by the VAX-Linker as an initial value.

All processes mapping Data Element Directories must use the Data Element Directory mapping context structure. The number of entries in this structure is identical to the number of entries in the Master Directory of the Data Base. If the Master Directory will be expanded (rebuilt with a larger size) the local mapping context structure must be allocated again, using the new number of entries. This value must be copied from the Master Directory structure element LM$MDEN_MASTER_DIR_MAX_ENTRIES to the mapping context structure REFER element LM$DBMC_MDIR_ENTRIES. These structures contain the identification version number, and the local virtual pointers for each Data Element Directory which was mapped, and the mapping access mode (read/write or read only). If the identification version number is zero or is not equal to the identification version numbers in the Main Directory the Data Element Directory must be mapped again.

In addition to each Data Element Directory a structure of the mapping context of all Data Elements possible in the Data Element Directory will be allocated. These structures contain the identification version number and the mapping access mode (read/write or read only). If the identification version number is zero or is not equal to the identification version numbers in the directory the Data Element data Pool must be mapped again.

All processes mapping Data Element Areas must use the Area mapping context structure. The number of entries in this structure is identical to the number of entries in the Area Directory of the Data Base. If the Area Directory will be expanded (rebuilt with a larger size) the local mapping context structure must be allocated again, using the new number of entries. This value must be copied from the Area Directory structure element LM$ADEN_AREA_DIR_MAX_ENTRIES to the mapping context structure REFER element LM$DBMC_AREA_ENTRIES. These structures contain the identification version number, and the local virtual pointers for each Area which was mapped, and the mapping access mode (read/write or read only). If the identification version number is zero or is not equal to the identification version numbers in the Area Directory the Area must be mapped again.

In addition to the mapping context information for each Area in a Data Base the structure SM$DBMC_MAIN includes separate information for the Home Block and for each main Directory (Area-, Pool-, Master-, Type-Directory) of a Data Base. This additional information allows a direct access to the main directories of the Data Base. The local mapping contexts of the directory Areas will only be held by these structures and not by the general Area context part of the SM$DBMC_MAIN structure. The structure SM$DBMC_MAIN includes also the I/O channel number of the opened Global Section file and its full Global Section name.

All structures will be allocated and the members of the structures will be filled during the attachment of a Data Base.

The PL/I structures of the local mapping context must be included by any procedure addressing parts of the Data Base,
GOOINC(SM$DBMC).

The PL/I structures have the following format:

```
/* GOOINC(SM$DBMC), Data Base Directory Mapping Context            */

/* Total number of Data Base allowed for attach                    */
/* = total number of dynamic lists allowed                         */

%REPLACE L_SM$ALL_DBMC_MAX_NUMBER BY 128;

/* Total number of different specific mapping pointers             */

%REPLACE L_SM$ALL_DUMMY_MAX_NUMBER BY 128;

                /* default base pointer to the Data Bases and      */
                /* dynamic lists mapping context pointers          */
                /* This pointer is an EXTERNAL !!!                 */
DCL     PM$DBMC                         POINTER EXTERNAL STATIC;

/* Pointers to the mapping context of:                             */
/*    all attached Data Bases                                      */
/*    all dynamic list structures used by one session.             */
/*    attached Display information                                 */
/* This array has L_SM$ALL_DBMC_MAX_NUMBER entries.
/* To get the mapping context of a Data Base, copy the             */
/* corresponding pointer P_SM$ALL_DBMC_MAIN(index) to the default  */
/* base pointer of the structure SM$DBMC.                          */
/* To get the mapping context of a dynamic list, copy the          */
/* corresponding pointer P_SM$ALL_DYNAMIC(index) to the default    */
/* base pointer of the dynamic list structure.                     */

DCL 1 SM$ALL_DBMC       BASED(PM$DBMC),

                /* base pointer to the Data Base mapping context   */
      2 P_SM$ALL_DBMC_MAIN(L_SM$ALL_DBMC_MAX_NUMBER)    POINTER,
                /* base pointer to the dynamic list structure      */
      2 P_SM$ALL_DYNAMIC(L_SM$ALL_DBMC_MAX_NUMBER)      POINTER,
                /* base pointer to the Display mapping context     */
      2 P_SD$DISPLAY_MAIN       POINTER,
```

```
                /* base pointer to other context                    */
      2 P_SM$DUMMY(L_SM$ALL_DUMMY_MAX_NUMBER - 1)       POINTER;


                /* default base pointer to a mapping context of     */
                /* a selected Data Base                             */
DCL     P_SM$DBMC_MAIN                  POINTER;


/* Home Block and Main Directory specific information              */


DCL 1 SM$DBMC_MAIN      BASED(P_SM$DBMC_MAIN),

                /* logical name of the Global Section (Data Base)   */
      2 CV_SM$GL_SEC_NAME               CHARACTER (254) VAR,
                /* Global Section creating and mapping flags        */
                /*  These flags are defined with INCLUDE $SECDEF    */
                /*  Default: SEC$M_GBL | SEC$M_PERM                 */
      2 B_SM$GL_SEC_FLAGS               BIT(32) ALIGNED,
                /* I/O channel number of Global Section File        */
      2 L_SM$CHAN                       BIN FIXED (31),
                /*Index of Data Base in structure of all Data Bases*/
      2 L_SM$DBMC_INDEX                 BIN FIXED (31),
                /* Lock identification of the whole Data Base       */
      2 L_SM$DBMC_DB_LOCK_ID            BIN FIXED (31),
                /* local version number of Home Block mapped        */
      2 L_SM$HBV1_HB_ID_VERSION         BIN FIXED (31),
                /* pointer array after mapping the Home Block       */
      2 P_SM$HBV1(2)                    POINTER,
                /* Flag word of Home Block mapped                   */
      2 S_SM$HBV1_FLAGS UNION,
        3 B_SM$HBV1_FLAGS                  BIT(32) ALIGNED,
        3 S_SM$HBV1_FLAG_BITS,
                /* Map Flag, '1'B: Write Access, '0'B: Read only    */
          4 B_SM$HBV1_FLAG_MAP_ACCESS           BIT(1),
          4 B_SM$HBV1_FLAG_REST                 BIT(31),
                /* Lock identification of the mapped Home Block     */
      2 L_SM$DBMC_HBV1_LOCK_ID          BIN FIXED (31),
                /* Lock identification of all Main Directories      */
                /*  i.e. Area, Pool and Master Directory            */
      2 L_SM$DBMC_MAIN_DIR_LOCK_ID      BIN FIXED (31),
```

```
          /* local version number of Area Directory mapped   */
2 L_SM$ADIR_AREA_ID_VERSION       BIN FIXED (31),
          /* pointer array after mapping the Area Directory  */
2 P_SM$ADIR(2)                    POINTER,
          /* pointer mapping the Area Directory entries      */
2 P_SM$ADEN                       POINTER,
          /* Flag word of Area Directory mapped              */
2 S_SM$ADIR_FLAGS UNION,
  3 B_SM$ADIR_FLAGS                  BIT(32) ALIGNED,
  3 S_SM$ADIR_FLAG_BITS,
          /* Map Flag, '1'B: Write Access, '0'B: Read only   */
    4 B_SM$ADIR_FLAG_MAP_ACCESS           BIT(1),
    4 B_SM$ADIR_FLAG_REST                 BIT(31),
          /* local version number of Pool Directory mapped   */
2 L_SM$PDIR_AREA_ID_VERSION       BIN FIXED (31),
          /* pointer array after mapping the Pool Directory  */
2 P_SM$PDIR(2)                    POINTER,
          /* pointer mapping the Pool Directory entries      */
2 P_SM$PDEN                       POINTER,
          /* Flag word of Pool Directory mapped              */
2 S_SM$PDIR_FLAGS UNION,
  3 B_SM$PDIR_FLAGS                  BIT(32) ALIGNED,
  3 S_SM$PDIR_FLAG_BITS,
          /* Map Flag, '1'B: Write Access, '0'B: Read only   */
    4 B_SM$PDIR_FLAG_MAP_ACCESS           BIT(1),
    4 B_SM$PDIR_FLAG_REST                 BIT(31),
          /* local version number of Master Directory mapped */
2 L_SM$MDIR_AREA_ID_VERSION       BIN FIXED (31),
          /* pointer array after mapping the Master Directory*/
2 P_SM$MDIR(2)                    POINTER,
          /* pointer mapping the Master Directory entries    */
2 P_SM$MDEN                       POINTER,
          /* Flag word of Master Directory mapped            */
2 S_SM$MDIR_FLAGS UNION,
  3 B_SM$MDIR_FLAGS                  BIT(32) ALIGNED,
  3 S_SM$MDIR_FLAG_BITS,
          /* Map Flag, '1'B: Write Access, '0'B: Read only   */
    4 B_SM$MDIR_FLAG_MAP_ACCESS           BIT(1),
    4 B_SM$MDIR_FLAG_REST                 BIT(31),
          /* local version number of Type Directory mapped   */
2 L_SM$TDIR_ID_VERSION            BIN FIXED (31),
          /* pointer array after mapping the Type Directory  */
```

```
      2 P_SM$TDIR(2)                    POINTER,
               /* pointer mapping the Type Directory entries      */
      2 P_SM$TDEN                       POINTER,
               /* Flag word of Type Directory mapped              */
      2 S_SM$TDIR_FLAGS UNION,
        3 B_SM$TDIR_FLAGS                  BIT(32) ALIGNED,
        3 S_SM$TDIR_FLAG_BITS,
          4 B_SM$TDIR_FLAG_REST                BIT(32),
               /* Type Directory index in the Master Directory    */
      2 L_SM$DBMC_TDIR_INDEX            BIN FIXED(31),
               /* Lock identification of the mapped Type Directory*/
      2 L_SM$DBMC_TDIR_LOCK_ID          BIN FIXED (31),
             /* pointer to the Type Data Element mapping structure */
      2 P_SM$DBMC_TDIR_DEMC             POINTER,
               /* default base pointer to the mapping context of  */
               /* all Data Element Directories of a selected      */
               /* Data Base                                       */
      2 P_SM$DBMC_EDIR                  POINTER,
               /* REFER element for Data Element Directory entries*/
      2 L_SM$DBMC_MDIR_ENTRIES          BIN FIXED (31),
               /* default base pointer to the mapping context of  */
               /* all Areas of a selected Data Base               */
      2 P_SM$DBMC_AREA                  POINTER,
               /* REFER element for Area Directory entries        */
      2 L_SM$DBMC_AREA_ENTRIES          BIN FIXED (31);


/*Mapping contexts for each Data Element Directory in the Data Base*/

DCL 1 SM$DBMC_EDIR      BASED(P_SM$DBMC_EDIR),

               /* number of Master Directory entries =            */
               /* number of mapping context entries               */
      2 LM$DBMC_MASTER_DIR_MAX_ENTRIES  BIN FIXED (31),

               /* start of entries                                */
      2 SM$DBMC_EDIR_ENTRY
      (L_SM$DBMC_MDIR_ENTRIES REFER(LM$DBMC_MASTER_DIR_MAX_ENTRIES)),
               /* local version number of an DE Directory mapped  */
        3 LM$DBMC_EDIR_ID_VERSION         BIN FIXED (31),
               /* pointer array after mapping the DE Directory    */
        3 PM$DBMC_EDIR(2)                 POINTER,
```

```
              /* pointer mapping the DE Directory entries        */
        3 PM$DBMC_EDEN                    POINTER,
              /* Flag word of DE Directory mapped                */
        3 S_SM$DBMC_EDIR_FLAGS  UNION,
          4 B_SM$DBMC_EDIR_FLAGS                BIT(32) ALIGNED,
          4 S_SM$DBMC_EDIR_FLAG_BITS,
            5 B_SM$DBMC_EDIR_FLAG_REST                  BIT(32),
              /* Lock identification of a mapped Data Element Dir*/
        3 LM$DBMC_EDIR_LOCK_ID            BIN FIXED (31),
              /* pointer to the Data Element mapping structure   */
        3 PM$DBMC_DEMC                    POINTER;


/*Mapping context for all Data Elements in a Data Element Directory*/

              /* default base pointer                            */
DCL     P_SM$DBMC_DE            POINTER;
              /* REFER element                                   */
DCL     L_SM$DBMC_DE_ENTRIES    BIN FIXED (31);

DCL 1 SM$DBMC_DE        BASED(P_SM$DBMC_DE),

              /* number of Data Element Directory entries =      */
              /* number of mapping context entries               */
      2 LM$DBMC_DE_DIR_MAX_ENTRIES      BIN FIXED (31),
              /* start of entries                                */
      2 SM$DBMC_DE_ENTRY
        (L_SM$DBMC_DE_ENTRIES REFER(LM$DBMC_DE_DIR_MAX_ENTRIES)),
              /* local version number of a mapped Data Element   */
        3 LM$DBMC_DE_ID_VERSION           BIN FIXED (31),
              /* Pool index of a mapped Data Element             */
        3 LM$DBMC_DE_POOL_INDEX           BIN FIXED (31),
              /* Lock identification of a mapped Data Element    */
        3 LM$DBMC_DE_LOCK_ID              BIN FIXED (31);


/* Mapping contexts for each Area in the Data Base               */

DCL 1 SM$DBMC_AREA      BASED(P_SM$DBMC_AREA),

              /* number of Area Directory entries =              */
              /* number of mapping context entries               */
```

```
      2 LM$DBMC_AREA_DIR_MAX_ENTRIES    BIN FIXED (31),
               /* start of entries                                  */
      2 SM$DBMC_AREA_ENTRY
        (L_SM$DBMC_AREA_ENTRIES REFER(LM$DBMC_AREA_DIR_MAX_ENTRIES)),
               /* local version number of an Area mapped            */
        3 LM$DBMC_AREA_ID_VERSION         BIN FIXED (31),
               /* pointer array after mapping the Area              */
        3 PM$DBMC_AREA(2)                 POINTER,
               /* Flag word of Area mapped                          */
        3 S_SM$DBMC_AREA_FLAGS  UNION,
          4 B_SM$DBMC_AREA_FLAGS                BIT(32) ALIGNED,
          4 S_SM$DBMC_AREA_FLAG_BITS,
               /* Map Flag, '1'B: Write Access, '0'B: Read only     */
            5 B_SM$DBMC_AREA_FLAG_MAP_ACCESS            BIT(1),
            5 B_SM$DBMC_AREA_FLAG_REST                  BIT(31),
               /* Lock identification of a mapped Area              */
        3 LM$DBMC_AREA_LOCK_ID            BIN FIXED (31);

/* Directory Extent types                                           */

               /* Name extent                                       */
%REPLACE       DEX__TYPE_NAME          BY      1;
               /* Link extent                                       */
%REPLACE       DEX__TYPE_LINK          BY      2;
               /* Data Element data descriptor extent               */
%REPLACE       DEX__TYPE_DESC          BY      3;
               /* Data Element name array descriptor                */
%REPLACE       DEX__TYPE_NAME_ARRAY    BY      4;
               /* Access rights descriptor                          */
%REPLACE       DEX__TYPE_ACCESS_RIGHTS BY      5;
```

C.2 Home Block

The very first data pages in each Data Base are called the Home Block.

The Home Block has no Data Base Area Header in front, but a 12 byte PL/I AREA information block for DEC & IBM internal usage. This allows the addressing of the Home Block as a PL/I AREA inclusive OFFSET addressing within the Home Block.

The fifth longword of the Home Block and with it of the whole Data Base is a version number of the Data Base structure. With this version the interpretation of the Data Base may be differentiated.

To get the structure into a procedure include: SM$HBV1 which will now be listed:

```
/* GOOINC(SM$HBV1), Home Block structure Version 1              */

               /* default base pointer P_SM$HBV1 is defined in */
               /* SM$DBMC                                      */

               /* REFER element                                */
DCL     L_SM$HBV1_BITMAP               BIN FIXED (31);
               /* Home Block Area                              */
DCL     A_SM$HBV1 AREA(LM$HBV1_LEN_HOME_BLOCK*512)
               BASED(P_SM$HBV1(1));

DCL 1 SM$HBV1   BASED(P_SM$HBV1(1)),
               /* reserved for DEC \& IBM Area information     */
      2 LM$HBV1_RESERVED(4)     BIN FIXED (31),
               /* version of Data Base                         */
      2 LM$HBV1_VERSION                BIN FIXED (31),
               /* identification version of Data Base          */
      2 LM$HBV1_HB_ID_VERSION          BIN FIXED (31),
               /* length of Home Block in pages (512 bytes)    */
      2 LM$HBV1_LEN_HOME_BLOCK         BIN FIXED (31),
               /* length of Data Base in pages (512 bytes)     */
      2 LM$HBV1_LEN_DATA_BASE          BIN FIXED (31),
               /* length of Area Directory in pages (512 bytes)*/
      2 LM$HBV1_AREA_DIR_LENGTH        BIN FIXED (31),
               /* start page of Area Directory in Data Base    */
      2 LM$HBV1_AREA_DIR_START_PAGE    BIN FIXED (31),
               /* length of Pool Directory in pages (512 bytes)*/
      2 LM$HBV1_POOL_DIR_LENGTH        BIN FIXED (31),
               /* start page of Pool Directory in Data Base    */
      2 LM$HBV1_POOL_DIR_START_PAGE    BIN FIXED (31),
               /*length of Master Directory in pages(512 bytes)*/
```

```
2 LM$HBV1_MASTER_DIR_LENGTH       BIN FIXED (31),
          /* start page of Master Directory in Data Base  */
2 LM$HBV1_MASTER_DIR_START_PAGE   BIN FIXED (31),
          /* OFFSET to Home Block extent                  */
2 OM$HBV1_EXTENT_OFF              OFFSET,
          /* creation time of Data Base file (Global Sec.)*/
2 CFM$HBV1_DB_FILE_CREATION_TIME  CHARACTER (24),
          /* creation time of Data Base as a Global Sect. */
2 CFM$HBV1_DB_LAST_CREATION_TIME  CHARACTER (24),
          /* deletion time of Data Base (Global Section)  */
2 CFM$HBV1_DB_LAST_DELETION_TIME  CHARACTER (24),
          /* last backup time of Data Base (Global Sect.) */
2 CFM$HBV1_DB_BACKUP_TIME         CHARACTER (24),
          /* current update number of Data Base           */
2 CFM$HBV1_DB_LAST_UPDATE_TIME    CHARACTER (24),
          /* name of Data Base                            */
2 CVM$HBV1_DB_NAME                CHARACTER (254) VAR,
          /* file name of Data Base (Global Section)      */
2 CVM$HBV1_DB_FILE_NAME           CHARACTER (254) VAR,
          /* number of bytes/bit in Bit Map               */
2 LM$HBV1_NBYTE_BIT               BIN FIXED (31),
          /* number of data byte clusters = pages         */
2 LM$HBV1_NDATA_BYTE_CLUSTERS     BIN FIXED (31),
          /* largest free contiguous part in Data Base    */
          /* in cluster units                             */
2 LM$HBV1_LARGEST_FREE            BIN FIXED (31),
          /* smallest free contiguous part in Data Base   */
          /* in cluster units                             */
2 LM$HBV1_SMALLEST_FREE           BIN FIXED (31),
          /* number of fragments in Data Base             */
2 LM$HBV1_NFRAGMENTS              BIN FIXED (31),
          /* Bit Map length in bytes                      */
2 LM$HBV1_BITMAP_LENGTH           BIN FIXED (31),
          /* Bit Map for the Data Base (1 Bit = 1 Page)   */
2 BM$HBV1_BITMAP
   BIT(L_SM$HBV1_BITMAP REFER(LM$HBV1_NDATA_BYTE_CLUSTERS));
```

C.3 Area Header

All Areas allocated in the Data Base must be preceded by an Area header of the following format.

An Area size must always be a multiple of 512 bytes (1 VAX page).

The Area itself is divided into fixed data clusters. The cluster size is defined by the initialization of the Area.

The Bit Map of each Area reflects the allocation of data clusters in the Area. For each data cluster one bit is reserved. If the data cluster is allocated the corresponding bit is set.

The Bit Map of the Area Header contains all allocation information about the whole Area incl. the header itself. Therefore the header has to be allocated as the first data block using the data cluster size.

To get the PL/I structure of an Area Header into a procedure include: SM$ARHD which will now be listed:

```
/* GOOINC(SM$ARHD), Area header                                    */

               /* default base pointer                             */
DCL     P_SM$ARHD               POINTER;
               /* REFER element                                    */
DCL     L_SM$ARHD_BITMAP        BIN FIXED (31);

DCL 1 SM$ARHD   BASED(P_SM$ARHD),
               /* reserved for DEC \& IBM Area information         */
      2 LM$ARHD_RESERVED(4)     BIN FIXED (31),
               /* total Area length in pages                       */
      2 LM$ARHD_AREA_LENGTH     BIN FIXED (31),
               /* Area head length in bytes                        */
      2 LM$ARHD_AREAHEAD_LENGTH BIN FIXED (31),
               /* Area id index longword                           */
      2 LM$ARHD_AREA_INDEX      BIN FIXED (31),
               /* Area id version word                             */
      2 LM$ARHD_AREA_ID_VERSION BIN FIXED (31),
               /* no. of bytes/bit in Bit Map                      */
               /* = data cluster size                              */
      2 LM$ARHD_NBYTE_BIT       BIN FIXED (31),
               /* data space length in bytes                       */
      2 LM$ARHD_DATA_LENGTH     BIN FIXED (31),
      2 SM$ARHD_TEMP_AREA_HEADER UNION,
               /* OFFSET to start of data                          */
        3 OM$ARHD_START_DATA            OFFSET,
               /* equiv. value of OFFSET                           */
        3 LM$ARHD_OSTARTDATA_VALUE      BIN FIXED (31),
```

```
          /* no. of data byte-blocks                        */
2 LM$ARHD_NDATA_BYTE_CLUSTERS BIN FIXED (31),
          /* largest free contiguous data                   */
          /* in cluster units                               */
2 LM$ARHD_LARGEST_FREE    BIN FIXED (31),
          /* smallest free contiguous data                  */
          /* in cluster units                               */
2 LM$ARHD_SMALLEST_FREE   BIN FIXED (31),
          /* number of fragments in data                    */
2 LM$ARHD_NFRAGMENTS      BIN FIXED (31),
          /* Bit Map length in bytes                        */
2 LM$ARHD_BITMAP_LENGTH   BIN FIXED (31),
          /* Bit Map                                        */
2 BM$ARHD_BITMAP
BIT(L_SM$ARHD_BITMAP REFER(LM$ARHD_NDATA_BYTE_CLUSTERS));
```

C.4 General Directory Format

A Directory has always a fixed part, the Directory Header, and then n entry blocks. Each entry block has a fixed length and depends on the Directory type. To get an entry block an Identification index (index longword) is used. This index can be used as an index to an array of a structure defining the Directory.

Each Directory entry has at least one extent part, the name of the entry. This extent part has a variable length and is allocated in a heap storage following the Directory. Each extent may have another extent and so on. This method allows to have variable length Directory entries.

To get the standard Directory Header PL/I structure into a procedure include: SM$DIRH with structures SM$DIRH, SM$DENT, SM$DIEX, and SM$DNEX which will now be listed:

```
/* GOOINC(SM$DIRH), Directory Header structure                    */

                /* default base pointer                           */
DCL     P_SM$DIRH               POINTER;
                /* REFER element                                  */
DCL     L_SM$DIRH_BITMAP        BIN FIXED (31);
                /* Directory Area                                 */
DCL     A_SM$DIRH AREA(LM$DIRH_AREA_LENGTH*512) BASED(P_SM$DIRH);

DCL 1 SM$DIRH   BASED(P_SM$DIRH),
                /* reserved for DEC \& IBM Area information       */
      2 LM$DIRH_RESERVED(4)             BIN FIXED (31),
                /* total Area length in pages                     */
      2 LM$DIRH_AREA_LENGTH             BIN FIXED (31),
                /* Area head length in bytes                      */
      2 LM$DIRH_AREAHEAD_LENGTH         BIN FIXED (31),
                /* Area id index longword                         */
      2 LM$DIRH_AREA_INDEX              BIN FIXED (31),
                /* Area id version word                           */
      2 LM$DIRH_AREA_ID_VERSION         BIN FIXED (31),
                /* no. of bytes/bit in Bit Map                    */
                /* = data cluster size                            */
      2 LM$DIRH_NBYTE_BIT               BIN FIXED (31),
                /* data space length in bytes                     */
      2 LM$DIRH_DATA_LENGTH             BIN FIXED (31),
      2 SM$DIRH_TEMP_AREA_HEADER UNION,
                /* OFFSET to start of data                        */
        3 OM$DIRH_START_DATA                    OFFSET,
                /* equiv. value of OFFSET                         */
        3 LM$DIRH_OSTARTDATA_VALUE              BIN FIXED (31),
```

```
                /* no. of data byte-blocks                        */
      2 LM$DIRH_NDATA_BYTE_CLUSTERS     BIN FIXED (31),
                /* largest free contiguous data                   */
                /* in cluster units                               */
      2 LM$DIRH_LARGEST_FREE            BIN FIXED (31),
                /* smallest free contiguous data                  */
                /* in cluster units                               */
      2 LM$DIRH_SMALLEST_FREE           BIN FIXED (31),
                /* number of fragments in data                    */
      2 LM$DIRH_NFRAGMENTS              BIN FIXED (31),
                /* Bit Map length in bytes                        */
      2 LM$DIRH_BITMAP_LENGTH           BIN FIXED (31),
                /* Bit Map                                        */
      2 BM$DIRH_BITMAP
      BIT(L_SM$DIRH_BITMAP REFER(LM$DIRH_NDATA_BYTE_CLUSTERS));

/* Directory Entry structure                                      */

                /* default base pointer                           */
DCL     P_SM$DENT               POINTER;
                /* REFER element                                  */
DCL     (L_SM$DENT_ENTRY,L_SM$DENT_DIR_VAR)     BIN FIXED (31);

DCL 1 SM$DENT    BASED(P_SM$DENT),
                /* maximum number of entries                      */
      2  LM$DENT_DIR_MAX_ENTRIES        BIN FIXED (31),
                /* current number of entries                      */
      2  LM$DENT_DIR_CURR_ENTRIES       BIN FIXED (31),
                /* length of the variable part of an entry        */
                /* in bytes                                       */
      2  LM$DENT_DIR_ENTRY_VAR_LENGTH   BIN FIXED (31),
                /* top index of binary name tree                  */
      2  LM$DENT_DIR_TOP_NAME_INDEX     BIN FIXED (31),
                /* start of entries                               */
      2  SM$DENT_ENTRY
      (L_SM$DENT_ENTRY REFER(LM$DENT_DIR_MAX_ENTRIES)),
                /* entry id version                               */
        3 LM$DENT_ID_VERSION                    BIN FIXED (31),
                /* Flags of Directory entry                       */
        3 SM$DENT_FLAGS UNION,
          4 BM$DENT_FLAGS                         BIT(32) ALIGNED,
          4 SM$DENT_FLAG_BITS,
```

```
            /*Entry usage flag, '1'B: in use, '0'B: not used*/
          5 BM$DENT_FLAG_ENTRY_IN_USE               BIT(1),
            /* Deletion protection flag, '1'B:do not delete */
            /*  directory entry without privilege          */
          5 BM$DENT_FLAG_PROTECTED                  BIT(1),
            /* Data Element entry queue flag               */
            /*  '1'B: entry is part of a queue             */
          5 BM$DENT_FLAG_QUEUE_MEMBER               BIT(1),
            /* Data Element entry queue header flag        */
            /*  '1'B: entry is the head of a queue         */
          5 BM$DENT_FLAG_QUEUE_HEAD                 BIT(1),
            /* Data Element entry name array flag          */
            /*  '1'B: entry is a name array member         */
          5 BM$DENT_FLAG_NAME_ARRAY_MEMBER          BIT(1),
            /* Data Element entry name array head flag     */
            /*  '1'B: entry is the head of a name array    */
          5 BM$DENT_FLAG_NAME_ARRAY_HEAD            BIT(1),
            /* Data Element Type Descriptor Directory      */
            /*  '1'B: entry is a Type Descriptor Directory */
          5 BM$DENT_FLAG_TYPE_DIR                   BIT(1),
            /* Not yet used bits                           */
          5 BM$DENT_FLAG_REST                       BIT(25),
            /* OFFSET to Directory extent                  */
        3 OM$DENT_EXTENT_OFF                  OFFSET,
            /* Directory index of preceding name           */
        3 LM$DENT_PRE_NAME_INDEX              BIN FIXED (31),
            /* Directory index of following name           */
        3 LM$DENT_FOLLOW_NAME_INDEX           BIN FIXED (31),
            /* binary name tree weight of this entry       */
        3 LM$DENT_NODE_WEIGHT                 BIN FIXED (31),
            /* individual part of entry (depen. on Dir.)   */
        3 HM$DENT_DIR_VAR
        (L_SM$DENT_DIR_VAR REFER(LM$DENT_DIR_ENTRY_VAR_LENGTH))
                                              BIN FIXED(7);

/* General Directory Extent structure                      */

            /* default base pointer                        */
DCL     P_SM$DIEX               POINTER;

DCL 1 SM$DIEX    BASED(P_SM$DIEX),
            /* Directory extent type                       */
```

```
      2  LM$DIEX_TYPE                   BIN FIXED (31),
                /* OFFSET to next Dir. extent for that entry    */
      2  OM$DIEX_NEXT_EXT               OFFSET;

/* Directory Name Extent structure                              */

                /* default base pointer                         */
DCL     P_SM$DNEX               POINTER;
                /* REFER element                                */
DCL     L_SM$DNEX_DIR_NAME      BIN FIXED (31);

DCL 1 SM$DNEX    BASED(P_SM$DNEX),
                /* Directory extent type                        */
      2  LM$DNEX_TYPE                   BIN FIXED (31),
                /* OFFSET to next Dir. extent for that entry    */
      2  OM$DNEX_NEXT_EXT               OFFSET,
                /* minimal name abbreviation in bytes           */
      2  IM$DNEX_MIN_ABBR               BIN FIXED (15),
                /* max. name length in bytes                    */
      2  IM$DNEX_MAX_LENGTH             BIN FIXED (15),
                /* name of entry                                */
      2  CVM$DNEX_DIR_NAME
      CHARACTER(L_SM$DNEX_DIR_NAME REFER(IM$DNEX_MAX_LENGTH)) VAR;
```

C.5 Area Directory

An Area is the smallest mapping unit for the Data Base. Several Areas are collected in a Pool. If there would be not enough room in an Area, a new, larger Area of the same Pool will be allocated.

The names of the Areas follow the Area Directory structure as Directory extents.

To get the PL/I structure into a procedure include: SM$ADIR with structures SM$ADIR and SM$ADEN which will now be listed:

```
/* GOOINC(SM$ADIR), Area Directory structure                    */

               /* maximum number of characters for Area name   */
%REPLACE       L_SM$ADIR_MAX_NAME_LENGTH       BY      254;

               /* Initial number of entries for Area Directory */
%REPLACE       L_SM$ADIR_INIT_ENTRIES          BY      512;

               /* default base pointer P_SM$ADIR is defined in */
               /* SM$DBMC                                      */
               /* REFER element                                */
DCL     L_SM$ADIR_BITMAP        BIN FIXED (31);
               /* Area Directory Area                          */
DCL     A_SM$ADIR AREA(LM$ADIR_DIR_AREA_LENGTH*512)
               BASED(P_SM$ADIR(1));

DCL 1 SM$ADIR    BASED(P_SM$ADIR(1)),
               /* reserved for DEC \& IBM Area information     */
      2  LM$ADIR_RESERVED(4)            BIN FIXED (31),
               /* total Area length in pages                   */
      2  LM$ADIR_DIR_AREA_LENGTH        BIN FIXED(31),
               /* Area header length in bytes                  */
               /*  incl. the Bit Map rounded to                */
               /*  data cluster                                */
      2  LM$ADIR_AREAHEAD_LENGTH        BIN FIXED(31),
               /* Area index longword                          */
      2  LM$ADIR_AREA_INDEX             BIN FIXED(31),
               /* Area id version                              */
      2  LM$ADIR_AREA_ID_VERSION        BIN FIXED(31),
               /* no.of bytes/bit in Bit Map                   */
      2  LM$ADIR_NBYTE_BIT              BIN FIXED(31),
               /* data space length in bytes                   */
               /*  without the Header                          */
      2  LM$ADIR_DATA_LENGTH            BIN FIXED(31),
```

```
                /* offset to data space                          */
      2  OM$ADIR_START_DATA             OFFSET,
                /* max. number of data clusters                  */
                /*  incl. the Header                             */
      2  LM$ADIR_NDATA_BYTE_CLUSTERS    BIN FIXED(31),
                /* largest free contiguous data                  */
                /* in cluster units                              */
      2  LM$ADIR_LARGEST_FREE           BIN FIXED (31),
                /* smallest free contiguous data                 */
                /* in cluster units                              */
      2  LM$ADIR_SMALLEST_FREE          BIN FIXED (31),
                /* number of fragments in data                   */
      2  LM$ADIR_NFRAGMENTS             BIN FIXED (31),
                /* Bit Map length in bytes                       */
      2  LM$ADIR_BITMAP_LENGTH          BIN FIXED(31),
                /* Bit Map                                       */
      2  BM$ADIR_BITMAP
      BIT(L_SM$ADIR_BITMAP REFER(LM$ADIR_NDATA_BYTE_CLUSTERS));

/* Area Directory Entry structure                                */

                /* default base pointer P_SM$ADEN is defined in */
                /* SM$DBMC                                       */
                /* REFER element                                 */
DCL     L_SM$ADEN_ENTRY         BIN FIXED (31);

DCL 1 SM$ADEN    BASED(P_SM$ADEN),
                /* maximum number of entries                     */
      2  LM$ADEN_AREA_DIR_MAX_ENTRIES   BIN FIXED (31),
                /* current number of entries                     */
      2  LM$ADEN_AREA_DIR_CURR_ENTRIES  BIN FIXED (31),
                /* length of the variable part of an entry       */
                /* in bytes                                      */
      2  LM$ADEN_DIR_ENTRY_VAR_LENGTH   BIN FIXED (31),
                /* top index of binary name tree                 */
      2  LM$ADEN_DIR_TOP_NAME_INDEX     BIN FIXED (31),
                /* start of entries                              */
      2  SM$ADEN_ENTRY
      (L_SM$ADEN_ENTRY REFER(LM$ADEN_AREA_DIR_MAX_ENTRIES)),
                /* entry id version                              */
        3 LM$ADEN_ID_VERSION                  BIN FIXED (31),
                /* Flags of Area Directory entry                 */
```

```
3 SM$ADEN_FLAGS UNION,
  4 BM$ADEN_FLAGS                        BIT(32) ALIGNED,
  4 SM$ADEN_FLAG_BITS,
       /*Entry usage flag, '1'B: in use, '0'b: not used*/
    5 BM$ADEN_FLAG_ENTRY_IN_USE               BIT(1),
       /* Deletion protection flag, '1'B:do not delete */
       /*  directory entry without privilege          */
    5 BM$ADEN_FLAG_PROTECTED                  BIT(1),
       /* Data Element entry queue flag               */
       /*  '1'B: entry is part of a queue             */
    5 BM$ADEN_FLAG_QUEUE_MEMBER               BIT(1),
       /* Data Element entry queue header flag        */
       /*  '1'B: entry is the head of a queue         */
    5 BM$ADEN_FLAG_QUEUE_HEAD                 BIT(1),
       /* Data Element entry name array flag          */
       /*  '1'B: entry is a name array member         */
    5 BM$ADEN_FLAG_NAME_ARRAY_MEMBER          BIT(1),
       /* Data Element entry name array head flag     */
       /*  '1'B: entry is the head of a name array    */
    5 BM$ADEN_FLAG_NAME_ARRAY_HEAD            BIT(1),
       /* Data Element Type Descriptor Directory      */
       /*  '1'B: entry is a Type Descriptor Directory */
    5 BM$ADEN_FLAG_TYPE_DIR                   BIT(1),
       /* Not yet used bits                           */
    5 BM$ADEN_FLAG_REST                       BIT(25),
       /* OFFSET to Directory extent                  */
3 OM$ADEN_EXTENT_OFF                   OFFSET,
       /* Area index of preceding name                */
3 LM$ADEN_PRE_NAME_INDEX               BIN FIXED (31),
       /* Area index of following name                */
3 LM$ADEN_FOLLOW_NAME_INDEX            BIN FIXED (31),
       /* binary name tree weight of this entry       */
3 LM$ADEN_NODE_WEIGHT                  BIN FIXED (31),
       /* length of Area in pages                     */
3 LM$ADEN_AREA_LENGTH                  BIN FIXED (31),
       /* start page of Area in the Data Base         */
3 LM$ADEN_START_PAGE                   BIN FIXED (31),
       /* number of bytes/bit in Area Bit Map         */
3 LM$ADEN_NBYTE_BIT                    BIN FIXED (31),
       /* index of Area Pool                          */
3 LM$ADEN_POOL_INDEX                   BIN FIXED (31),
       /* index of next Area in Pool                  */
```

```
   3 LM$ADEN_NEXT_POOL_AREA_INDEX         BIN FIXED (31),

         /* start of Directory extents                    */
2  LM$ADEN_FIRST_EXTENT           BIN FIXED (31);
```

C.6 Pool Directory

A Pool is a collection of several Areas with corresponding mapping attributes. The Areas collected in a Pool are linked together by their Area-index longwords. The index of the next Area in a link is kept in the Area Directory structure.

The names of the Pools follow the Pool Directory structure as Directory extents.

To get the PL/I structure into a procedure include: SM$PDIR with structures SM$PDIR and SM$PDEN which will now be listed:

```
/* GOOINC(SM$PDIR), Pool Directory structure                    */

                /* maximum number of characters for Pool name   */
%REPLACE        L_SM$PDIR_MAX_NAME_LENGTH       BY      254;

                /* Initial number of entries for Pool Directory */
%REPLACE        L_SM$PDIR_INIT_ENTRIES          BY      512;

                /* default base pointer P_SM$PDIR is defined in */
                /* SM$DBMC                                      */
                /* REFER element                                */
DCL     L_SM$PDIR_BITMAP        BIN FIXED (31);
                /* Pool Directory Area                          */
DCL     A_SM$PDIR AREA(LM$PDIR_DIR_AREA_LENGTH*512)
                BASED(P_SM$PDIR(1));

DCL 1 SM$PDIR    BASED(P_SM$PDIR(1)),
                /* reserved for DEC \& IBM Area information     */
      2  LM$PDIR_RESERVED(4)            BIN FIXED (31),
                /* total Area length in pages                   */
      2  LM$PDIR_DIR_AREA_LENGTH        BIN FIXED(31),
                /* Area header length in bytes                  */
                /*  incl. the Bit Map rounded to                */
                /*  data cluster                                */
      2  LM$PDIR_AREAHEAD_LENGTH        BIN FIXED(31),
                /* Area index longword                          */
      2  LM$PDIR_AREA_INDEX             BIN FIXED(31),
                /* Area id version                              */
      2  LM$PDIR_AREA_ID_VERSION        BIN FIXED(31),
                /* no.of bytes/bit in Bit Map                   */
      2  LM$PDIR_NBYTE_BIT              BIN FIXED(31),
                /* data space length in bytes                   */
                /*  without the Header                          */
```

```
      2  LM$PDIR_DATA_LENGTH            BIN FIXED(31),
                /* offset to data space                          */
      2  OM$PDIR_START_DATA             OFFSET,
                /* max. number of data clusters                  */
                /*  incl. the Header                             */
      2  LM$PDIR_NDATA_BYTE_CLUSTERS    BIN FIXED(31),
                /* largest free contiguous data                  */
                /* in cluster units                              */
      2  LM$PDIR_LARGEST_FREE           BIN FIXED (31),
                /* smallest free contiguous data                 */
                /* in cluster units                              */
      2  LM$PDIR_SMALLEST_FREE          BIN FIXED (31),
                /* number of fragments in data                   */
      2  LM$PDIR_NFRAGMENTS             BIN FIXED (31),
                /* Bit Map length in bytes                       */
      2  LM$PDIR_BITMAP_LENGTH          BIN FIXED(31),
                /* Bit Map                                       */
      2  BM$PDIR_BITMAP
      BIT(L_SM$PDIR_BITMAP REFER(LM$PDIR_NDATA_BYTE_CLUSTERS));

/* Pool Directory Entry structure                                */

                /* default base pointer P_SM$PDEN is defined in */
                /* SM$DBMC                                       */
                /* REFER element                                 */
DCL     L_SM$PDEN_ENTRY         BIN FIXED (31);

DCL 1 SM$PDEN    BASED(P_SM$PDEN),
                /* maximum number of entries                     */
      2  LM$PDEN_POOL_DIR_MAX_ENTRIES   BIN FIXED (31),
                /* current number of entries                     */
      2  LM$PDEN_POOL_DIR_CURR_ENTRIES  BIN FIXED (31),
                /* length of the variable part of an entry       */
                /* in bytes                                      */
      2  LM$PDEN_DIR_ENTRY_VAR_LENGTH   BIN FIXED (31),
                /* top index of binary name tree                 */
      2  LM$PDEN_DIR_TOP_NAME_INDEX     BIN FIXED (31),
                /* start of entries                              */
      2  SM$PDEN_ENTRY
      (L_SM$PDEN_ENTRY REFER(LM$PDEN_POOL_DIR_MAX_ENTRIES)),
                /* entry id version                              */
        3 LM$PDEN_ID_VERSION                  BIN FIXED (31),
```

```
      /* Flags of Pool Directory entry                */
3 SM$PDEN_FLAGS UNION,
  4 BM$PDEN_FLAGS                       BIT(32) ALIGNED,
  4 SM$PDEN_FLAG_BITS,
      /*Entry usage flag, '1'B: in use, '0'b: not used*/
    5 BM$PDEN_FLAG_ENTRY_IN_USE               BIT(1),
      /* Deletion protection flag, '1'B:do not delete */
      /*  directory entry without privilege          */
    5 BM$PDEN_FLAG_PROTECTED                  BIT(1),
      /* Data Element entry queue flag                */
      /*  '1'B: entry is part of a queue              */
    5 BM$PDEN_FLAG_QUEUE_MEMBER               BIT(1),
      /* Data Element entry queue header flag         */
      /*  '1'B: entry is the head of a queue          */
    5 BM$PDEN_FLAG_QUEUE_HEAD                 BIT(1),
      /* Data Element entry name array flag           */
      /*  '1'B: entry is a name array member          */
    5 BM$PDEN_FLAG_NAME_ARRAY_MEMBER          BIT(1),
      /* Data Element entry name array head flag      */
      /*  '1'B: entry is the head of a name array     */
    5 BM$PDEN_FLAG_NAME_ARRAY_HEAD            BIT(1),
      /* Data Element Type Descriptor Directory       */
      /*  '1'B: entry is a Type Descriptor Directory  */
    5 BM$PDEN_FLAG_TYPE_DIR                   BIT(1),
      /* Not yet used bits                            */
    5 BM$PDEN_FLAG_REST                       BIT(25),
      /* OFFSET to Directory extent                   */
3 OM$PDEN_EXTENT_OFF                  OFFSET,
      /* Pool index of preceding name                 */
3 LM$PDEN_PRE_NAME_INDEX              BIN FIXED (31),
      /* Pool index of following name                 */
3 LM$PDEN_FOLLOW_NAME_INDEX           BIN FIXED (31),
      /* binary name tree weight of this entry        */
3 LM$PDEN_NODE_WEIGHT                 BIN FIXED (31),
      /* minimum data size of Area in Pool in bytes   */
3 LM$PDEN_AREA_MIN_SIZE               BIN FIXED (31),
      /* index of first Area in Pool                  */
3 LM$PDEN_FIRST_POOL_AREA_INDEX       BIN FIXED (31),
      /* Highest number of Area in Pool               */
3 LM$PDEN_MAX_AREA_NUMBER             BIN FIXED (31),

      /* start of Directory extents                   */
```

```
2  LM$PDEN_FIRST_EXTENT          BIN FIXED (31);
```

C.7 Master Directory

A Data Element Directory is a collection of Data Elements. The name of each Data Element of a Data Base must be preceded by its Directory name. Each Data Element Directory has an entry in the Master Directory of a Data Base.

The names of the Directories follow the Master Directory structure as Directory extents.

To get the PL/I structure into a procedure include: SM$MDIR with structures SM$MDIR and SM$MDEN which will now be listed:

```
/* GOOINC(SM$MDIR), Master Directory structure                  */

             /* maximum number of characters for Directory name */
%REPLACE        L_SM$MDIR_MAX_NAME_LENGTH       BY      254;

             /* Initial number of entries for Master Directory  */
%REPLACE        L_SM$MDIR_INIT_ENTRIES          BY      512;

                /* default base pointer P_SM$MDIR is defined in */
                /* SM$DBMC                                      */
                /* REFER element                                */
DCL     L_SM$MDIR_BITMAP        BIN FIXED (31);
                /* Master Directory Area                        */
DCL     A_SM$MDIR AREA(LM$MDIR_DIR_AREA_LENGTH*512)
                BASED(P_SM$MDIR(1));

DCL 1 SM$MDIR    BASED(P_SM$MDIR(1)),
                /* reserved for DEC \& IBM Area information     */
      2  LM$MDIR_RESERVED(4)            BIN FIXED (31),
                /* total Area length in pages                   */
      2  LM$MDIR_DIR_AREA_LENGTH        BIN FIXED(31),
                /* Area header length in bytes                  */
                /*  incl. the Bit Map rounded to                */
                /*  data cluster                                */
      2  LM$MDIR_AREAHEAD_LENGTH        BIN FIXED(31),
                /* Area index longword                          */
      2  LM$MDIR_AREA_INDEX             BIN FIXED(31),
                /* Area id version                              */
      2  LM$MDIR_AREA_ID_VERSION        BIN FIXED(31),
                /* no.of bytes/bit in Bit Map                   */
      2  LM$MDIR_NBYTE_BIT              BIN FIXED(31),
                /* data space length in bytes                   */
                /*  without the Header                          */
```

```
      2  LM$MDIR_DATA_LENGTH            BIN FIXED(31),
                /* offset to data space                          */
      2  OM$MDIR_START_DATA             OFFSET,
                /* max. number of data clusters                  */
                /*  incl. the Header                             */
      2  LM$MDIR_NDATA_BYTE_CLUSTERS    BIN FIXED(31),
                /* largest free contiguous data                  */
                /* in cluster units                              */
      2  LM$MDIR_LARGEST_FREE           BIN FIXED (31),
                /* smallest free contiguous data                 */
                /* in cluster units                              */
      2  LM$MDIR_SMALLEST_FREE          BIN FIXED (31),
                /* number of fragments in data                   */
      2  LM$MDIR_NFRAGMENTS             BIN FIXED (31),
                /* Bit Map length in bytes                       */
      2  LM$MDIR_BITMAP_LENGTH          BIN FIXED(31),
                /* Bit Map                                       */
      2  BM$MDIR_BITMAP
      BIT(L_SM$MDIR_BITMAP REFER(LM$MDIR_NDATA_BYTE_CLUSTERS));

/* Master Directory Entry structure                              */

                /* default base pointer P_SM$MDEN is defined in */
                /* SM$DBMC                                       */
                /* REFER element                                 */
DCL     L_SM$MDEN_ENTRY         BIN FIXED (31);

DCL 1 SM$MDEN    BASED(P_SM$MDEN),
                /* maximum number of entries                     */
      2  LM$MDEN_MASTER_DIR_MAX_ENTRIES BIN FIXED (31),
                /* current number of entries                     */
      2  LM$MDEN_MASTER_DIR_CURR_ENTRIES BIN FIXED (31),
                /* length of the variable part of an entry       */
                /* in bytes                                      */
      2  LM$MDEN_DIR_ENTRY_VAR_LENGTH   BIN FIXED (31),
                /* top index of binary name tree                 */
      2  LM$MDEN_DIR_TOP_NAME_INDEX     BIN FIXED (31),
                /* start of entries                              */
      2  SM$MDEN_ENTRY
      (L_SM$MDEN_ENTRY REFER(LM$MDEN_MASTER_DIR_MAX_ENTRIES)),
                /* entry id version                              */
        3 LM$MDEN_ID_VERSION                  BIN FIXED (31),
```

```
        /* Flags of Master Directory entry              */
    3 SM$MDEN_FLAGS UNION,
      4 BM$MDEN_FLAGS                       BIT(32) ALIGNED,
      4 SM$MDEN_FLAG_BITS,
        /*Entry usage flag, '1'B: in use, '0'b: not used*/
        5 BM$MDEN_FLAG_ENTRY_IN_USE             BIT(1),
        /* Deletion protection flag, '1'B:do not delete */
        /*  directory entry without privilege          */
        5 BM$MDEN_FLAG_PROTECTED                BIT(1),
        /* Data Element entry queue flag               */
        /*  '1'B: entry is part of a queue             */
        5 BM$MDEN_FLAG_QUEUE_MEMBER             BIT(1),
        /* Data Element entry queue header flag        */
        /*  '1'B: entry is the head of a queue         */
        5 BM$MDEN_FLAG_QUEUE_HEAD               BIT(1),
        /* Data Element entry name array flag          */
        /*  '1'B: entry is a name array member         */
        5 BM$MDEN_FLAG_NAME_ARRAY_MEMBER        BIT(1),
        /* Data Element entry name array head flag     */
        /*  '1'B: entry is the head of a name array    */
        5 BM$MDEN_FLAG_NAME_ARRAY_HEAD          BIT(1),
        /* Data Element Type Descriptor Directory      */
        /*  '1'B: entry is a Type Descriptor Directory */
        5 BM$MDEN_FLAG_TYPE_DIR                 BIT(1),
        /* Not yet used bits                           */
        5 BM$MDEN_FLAG_REST                     BIT(25),
        /* OFFSET to Directory extent                  */
    3 OM$MDEN_EXTENT_OFF                OFFSET,
        /* Directory index of preceding name           */
    3 LM$MDEN_PRE_NAME_INDEX            BIN FIXED (31),
        /* Directory index of following name           */
    3 LM$MDEN_FOLLOW_NAME_INDEX         BIN FIXED (31),
        /* binary name tree weight of this entry       */
    3 LM$MDEN_NODE_WEIGHT               BIN FIXED (31),
        /* Area index of Data Element Directory        */
    3 LM$MDEN_EDIR_AREA_INDEX           BIN FIXED (31),

        /* start of Directory extents                  */
  2 LM$MDEN_FIRST_EXTENT          BIN FIXED (31);
```

C.8 Data Element Directory

Each Data Element has an entry in one Data Element Directory of a Data Base. Each Data Element Directory has one entry in the Master Directory of a Data Base.

The names of the Data Elements follow the Data Element Directory structure as Directory extents.

To get the PL/I structure into a procedure include: SM$EDIR with structures SM$EDIR, SM$EDEN, SM$EDDE, SM$EDNA, and SM$EDLE which will now be listed:

```
/* GOOINC(SM$EDIR), Data Element Directory structure           */

          /* maximum number of characters for Data Element name */
%REPLACE        L_SM$EDIR_MAX_NAME_LENGTH       BY      254;

                /* Initial number of entries for               */
                /*  Data Element Directory                     */
%REPLACE        L_SM$EDIR_INIT_ENTRIES          BY      512;

                /* default base pointer                        */
DCL     P_SM$EDIR(2)            POINTER;
                /* REFER element                               */
DCL     L_SM$EDIR_BITMAP        BIN FIXED (31);
                /* Data Element Directory Area                 */
DCL     A_SM$EDIR AREA(LM$EDIR_DIR_AREA_LENGTH*512)
                BASED(P_SM$EDIR(1));

DCL 1 SM$EDIR    BASED(P_SM$EDIR(1)),
                /* reserved for DEC \& IBM                     */
      2  LM$EDIR_RESERVED(4)           BIN FIXED (31),
                /* total Area length in pages                  */
      2  LM$EDIR_DIR_AREA_LENGTH       BIN FIXED(31),
                /* Area header length in bytes                 */
                /*  incl. the Bit Map rounded to               */
                /*  data cluster                               */
      2  LM$EDIR_AREAHEAD_LENGTH       BIN FIXED(31),
                /* Area index longword                         */
      2  LM$EDIR_AREA_INDEX            BIN FIXED(31),
                /* Area id version                             */
      2  LM$EDIR_AREA_ID_VERSION       BIN FIXED(31),
                /* no.of bytes/bit in Bit Map                  */
      2  LM$EDIR_NBYTE_BIT             BIN FIXED(31),
                /* data space length in bytes                  */
```

```
                /*  without the Header                          */
      2  LM$EDIR_DATA_LENGTH            BIN FIXED(31),
                /* offset to data space                        */
      2  OM$EDIR_START_DATA             OFFSET,
                /* max. number of data clusters                */
                /*  incl. the Header                           */
      2  LM$EDIR_NDATA_BYTE_CLUSTERS    BIN FIXED(31),
                /* largest free contiguous data                */
                /* in cluster units                            */
      2  LM$EDIR_LARGEST_FREE           BIN FIXED (31),
                /* smallest free contiguous data               */
                /* in cluster units                            */
      2  LM$EDIR_SMALLEST_FREE          BIN FIXED (31),
                /* number of fragments in data                 */
      2  LM$EDIR_NFRAGMENTS             BIN FIXED (31),
                /* Bit Map length in bytes                     */
      2  LM$EDIR_BITMAP_LENGTH          BIN FIXED(31),
                /* Bit Map                                     */
      2  BM$EDIR_BITMAP
         BIT(L_SM$EDIR_BITMAP REFER(LM$EDIR_NDATA_BYTE_CLUSTERS));

/* Data Element Directory Entry structure                      */

                /* default base pointer                        */
DCL     P_SM$EDEN               POINTER;
                /* REFER element                               */
DCL     L_SM$EDEN_ENTRY         BIN FIXED (31);

DCL 1 SM$EDEN    BASED(P_SM$EDEN),
                /* maximum number of entries                   */
      2  LM$EDEN_DE_DIR_MAX_ENTRIES     BIN FIXED (31),
                /* current number of entries                   */
      2  LM$EDEN_DE_DIR_CURR_ENTRIES    BIN FIXED (31),
                /* length of the variable part of an entry     */
                /* in bytes                                    */
      2  LM$EDEN_DIR_ENTRY_VAR_LENGTH   BIN FIXED (31),
                /* top index of binary name tree               */
      2  LM$EDEN_DIR_TOP_NAME_INDEX     BIN FIXED (31),
                /* start of entries                            */
      2  SM$EDEN_ENTRY
         (L_SM$EDEN_ENTRY REFER(LM$EDEN_DE_DIR_MAX_ENTRIES)),
                /* entry id version                            */
```

```
3 LM$EDEN_ID_VERSION                    BIN FIXED (31),
      /* Flags of Data Element Directory entry       */
3 SM$EDEN_FLAGS UNION,
  4 BM$EDEN_FLAGS                        BIT(32) ALIGNED,
  4 SM$EDEN_FLAG_BITS,
      /*Entry usage flag, '1'B: in use, '0'b: not used*/
    5 BM$EDEN_FLAG_ENTRY_IN_USE               BIT(1),
      /* Deletion protection flag, '1'B:do not delete */
      /*  directory entry without privilege          */
    5 BM$EDEN_FLAG_PROTECTED                  BIT(1),
      /* Data Element entry queue flag               */
      /*  '1'B: entry is part of a queue             */
    5 BM$EDEN_FLAG_QUEUE_MEMBER               BIT(1),
      /* Data Element entry queue header flag        */
      /*  '1'B: entry is the head of a queue         */
    5 BM$EDEN_FLAG_QUEUE_HEAD                 BIT(1),
      /* Data Element entry name array flag          */
      /*  '1'B: entry is a name array member         */
    5 BM$EDEN_FLAG_NAME_ARRAY_MEMBER          BIT(1),
      /* Data Element entry name array head flag     */
      /*  '1'B: entry is the head of a name array    */
    5 BM$EDEN_FLAG_NAME_ARRAY_HEAD            BIT(1),
      /* Data Element Type Descriptor Directory      */
      /*  '1'B: entry is a Type Descriptor Directory */
    5 BM$EDEN_FLAG_TYPE_DIR                   BIT(1),
      /* Not yet used bits                           */
    5 BM$EDEN_FLAG_REST                       BIT(25),
      /* OFFSET to Directory extent                  */
3 OM$EDEN_EXTENT_OFF                    OFFSET,
      /* Data Element index of preceding name        */
3 LM$EDEN_PRE_NAME_INDEX                BIN FIXED (31),
      /* Data Element index of following name        */
3 LM$EDEN_FOLLOW_NAME_INDEX             BIN FIXED (31),
      /* binary name tree weight of this entry       */
3 LM$EDEN_NODE_WEIGHT                   BIN FIXED (31),
      /* index of data Area of Data Element          */
3 LM$EDEN_DE_AREA_INDEX                 BIN FIXED (31),
      /* OFFSET to data in Area of Data Element      */
3 OM$EDEN_DE_START_DATA                 OFFSET,
      /* length of Data Element in bytes             */
3 LM$EDEN_DE_LENGTH                     BIN FIXED (31),
      /* index of Type Descriptor of Data Element    */
```

```
        3 LM$EDEN_DE_TYPE_INDEX                BIN FIXED (31),
              /* Data Descriptor structure like VMS          */
        3 SM$EDEN_DE_DESC UNION,
              /* Data Descriptor longword                    */
          4 LM$EDEN_DE_DESC                    BIN FIXED (31),
              /* Data Descriptor parts                       */
          4 SM$EDEN_DE_DESC_PARTS,
              /* Data Descriptor, Length of data item in bytes*/
              /* length in bits for bit,                     */
              /* length of an array element for arrays       */
            5 IM$EDEN_DE_DESC_LENGTH           BIN FIXED (15),
              /* Data Descriptor, data type code             */
            5 HM$EDEN_DE_DESC_DTYPE            BIN FIXED (7),
              /* Data Descriptor, data class code            */
            5 HM$EDEN_DE_DESC_CLASS            BIN FIXED (7),
              /* OFFSET to Data Element Directory Link extent */
        3 OM$EDEN_LINK_EXTENT_OFF              OFFSET,
              /* index of forward queued Data Element        */
        3 LM$EDEN_FORWARD_QUEUE_INDEX          BIN FIXED (31),
              /* index of backward queued Data Element       */
        3 LM$EDEN_BACKWARD_QUEUE_INDEX         BIN FIXED (31),
              /* index of Data Element name array head       */
        3 LM$EDEN_NAME_ARRAY_HEAD_INDEX        BIN FIXED (31),

              /* start of Directory extents                  */
      2  LM$EDEN_FIRST_EXTENT           BIN FIXED (31);

/* Data Element Directory Data Descriptor Extent structure (VMS)*/

              /* default base pointer                        */
DCL     P_SM$EDDE               POINTER;
              /* REFER element                               */
DCL     L_SM$EDDE_DIMCT         BIN FIXED (31);

DCL 1 SM$EDDE    BASED(P_SM$EDDE),
              /* Directory extent type                       */
      2  LM$EDDE_TYPE                   BIN FIXED (31),
              /* OFFSET to next Data Element Directory extent */
              /*  for that entry                             */
      2  OM$EDDE_NEXT_EXT               OFFSET,
              /* Array descriptor structure                  */
      2  SM$EDDE_ARRAY_DESC UNION,
```

```
        3 LM$EDDE_ARRAY_DESC          BIN FIXED (31),
        3 SM$EDDE_ARRAY_DESC_PARTS,
              /*  Scale factor                              */
          4 HM$EDDE_ARRAY_SCALE              BIN FIXED(7),
              /*  Number of decimal digits                  */
          4 HM$EDDE_ARRAY_DIGITS             BIN FIXED(7),
              /*  Array flag bits                           */
          4 HM$EDDE_ARRAY_AFLAGS             BIN FIXED(7),
              /*  Number of dimensions                      */
          4 HM$EDDE_ARRAY_DIMCT              BIN FIXED(7),
              /* Array bounds for each dimension            */
      2  SM$EDDE_ARRAY_BOUNDS
         (L_SM$EDDE_DIMCT REFER(HM$EDDE_ARRAY_DIMCT)),
              /* Lower bound (signed) of ith dimension      */
        3  LM$EDDE_ARRAY_LOWER         BIN FIXED (31),
              /* Upper bound (signed) of ith dimension      */
        3  LM$EDDE_ARRAY_UPPER         BIN FIXED (31);


/* Data Element Directory Name Array Descriptor Extent Structure*/

              /* default base pointer                       */
DCL     P_SM$EDNA              POINTER;
              /* REFER element                              */
DCL     L_SM$EDNA_DIMCT        BIN FIXED (31);

DCL 1 SM$EDNA    BASED(P_SM$EDNA),
              /* Directory extent type                      */
      2  LM$EDNA_TYPE                 BIN FIXED (31),
              /* OFFSET to next Data Element Directory Extent */
              /*  for that entry                            */
      2  OM$EDNA_NEXT_EXT              OFFSET,
              /* Name array descriptor structure            */
              /*  Name array flag bits                      */
      2 IM$EDNA_ARRAY_FLAGS            BIN FIXED (15),
              /*  Number of dimensions                      */
      2 IM$EDNA_ARRAY_DIMCT            BIN FIXED (15),
              /* Name array bounds for each dimension       */
      2  SM$EDNA_ARRAY_BOUNDS
         (L_SM$EDNA_DIMCT REFER(IM$EDNA_ARRAY_DIMCT)),
              /* Lower bound (signed) of ith dimension      */
        3  LM$EDNA_ARRAY_LOWER         BIN FIXED (31),
```

```
                /* Upper bound (signed) of ith dimension        */
        3  LM$EDNA_ARRAY_UPPER          BIN FIXED (31);

/* Data Element Directory Link Extent structure                 */

                /* default base pointer                         */
DCL     P_SM$EDLE               POINTER;

DCL 1 SM$EDLE    BASED(P_SM$EDLE),
                /* Directory extent type                        */
      2  LM$EDLE_TYPE                   BIN FIXED (31),
                /* OFFSET to next Data Element Dir. Link extent */
                /*  for that entry                              */
      2  OM$EDLE_NEXT_EDLE              OFFSET,
                /* Flags of Data Element Directory Link         */
      2  SM$EDLE_FLAGS UNION,
         3 LM$EDLE_FLAGS                        BIN FIXED (31),
         3 SM$EDLE_FLAG_BITS,
                /* Flag for Link direction of Data Element      */
                /*  forward : '1'B, backward : '0'B             */
           4 BM$EDLE_FLAG_DIRECTION                    BIT(1),
           4 BM$EDLE_FLAG_REST                         BIT(31),
                /* index of Directory of Linked Data Element    */
      2  LM$EDLE_LINKED_DIR_INDEX       BIN FIXED (31),
                /* index of Linked Data Element                 */
      2  LM$EDLE_LINKED_DE_INDEX        BIN FIXED (31);
```

C.9 Type Directory

The Types are Data Elements of the Directory $TYPE. The Type Directory is therefore identical to the Data Element Directory $TYPE.

The names of the Types follow the Type Directory structure as Directory extents.

To get the PL/I structure into a procedure include: SM$TDIR with structures SM$TDIR and SM$TDEN which will now be listed:

```
/* GOOINC(SM$TDIR), Data Type (Element) Directory structure     */

          /* maximum number of characters for Data Type name    */
%REPLACE        L_SM$TDIR_MAX_NAME_LENGTH       BY      254;

                /* Initial number of entries for                */
                /*  Data Type Directory                         */
%REPLACE        L_SM$TDIR_INIT_ENTRIES          BY      512;

                /* default base pointer P_SM$TDIR is defined in */
                /* SM$DBMC                                      */
                /* REFER element                                */
DCL     L_SM$TDIR_BITMAP        BIN FIXED (31);
                /* Data Type Directory Area                     */
DCL     A_SM$TDIR AREA(LM$TDIR_DIR_AREA_LENGTH*512)
                BASED(P_SM$TDIR(1));

DCL 1 SM$TDIR    BASED(P_SM$TDIR(1)),
                /* reserved for DEC \& IBM Area information     */
      2  LM$TDIR_RESERVED(4)            BIN FIXED (31),
                /* total Area length in pages                   */
      2  LM$TDIR_DIR_AREA_LENGTH        BIN FIXED(31),
                /* Area header length in bytes                  */
                /*  incl. the Bit Map rounded to                */
                /*  data cluster                                */
      2  LM$TDIR_AREAHEAD_LENGTH        BIN FIXED(31),
                /* Area index longword                          */
      2  LM$TDIR_AREA_INDEX             BIN FIXED(31),
                /* Area id version                              */
      2  LM$TDIR_AREA_ID_VERSION        BIN FIXED(31),
                /* no.of bytes/bit in Bit Map                   */
      2  LM$TDIR_NBYTE_BIT              BIN FIXED(31),
                /* data space length in bytes                   */
                /*  without the Header                          */
```

```
      2  LM$TDIR_DATA_LENGTH            BIN FIXED(31),
                /* offset to data space                          */
      2  OM$TDIR_START_DATA             OFFSET,
                /* max. number of data clusters                  */
                /*  incl. the Header                             */
      2  LM$TDIR_NDATA_BYTE_CLUSTERS    BIN FIXED(31),
                /* largest free contiguous data                  */
                /* in cluster units                              */
      2  LM$TDIR_LARGEST_FREE           BIN FIXED (31),
                /* smallest free contiguous data                 */
                /* in cluster units                              */
      2  LM$TDIR_SMALLEST_FREE          BIN FIXED (31),
                /* number of fragments in data                   */
      2  LM$TDIR_NFRAGMENTS             BIN FIXED (31),
                /* Bit Map length in bytes                       */
      2  LM$TDIR_BITMAP_LENGTH          BIN FIXED(31),
                /* Bit Map                                       */
      2  BM$TDIR_BITMAP
         BIT(L_SM$TDIR_BITMAP REFER(LM$TDIR_NDATA_BYTE_CLUSTERS));

/* Data Type (Element) Directory Entry structure                 */

                /* default base pointer P_SM$TDEN is defined in */
                /* SM$DBMC                                       */
                /* REFER element                                 */
DCL     L_SM$TDEN_ENTRY         BIN FIXED (31);

DCL 1 SM$TDEN    BASED(P_SM$TDEN),
                /* maximum number of entries                     */
      2  LM$TDEN_DE_DIR_MAX_ENTRIES     BIN FIXED (31),
                /* current number of entries                     */
      2  LM$TDEN_DE_DIR_CURR_ENTRIES    BIN FIXED (31),
                /* length of the variable part of an entry       */
                /* in bytes                                      */
      2  LM$TDEN_DIR_ENTRY_VAR_LENGTH   BIN FIXED (31),
                /* top index of binary name tree                 */
      2  LM$TDEN_DIR_TOP_NAME_INDEX     BIN FIXED (31),
                /* start of entries                              */
      2  SM$TDEN_ENTRY
         (L_SM$TDEN_ENTRY REFER(LM$TDEN_DE_DIR_MAX_ENTRIES)),
                /* entry id version                              */
         3 LM$TDEN_ID_VERSION                  BIN FIXED (31),
```

```
      /* Flags of Data Type Directory entry          */
3 SM$TDEN_FLAGS UNION,
  4 BM$TDEN_FLAGS                      BIT(32) ALIGNED,
  4 SM$TDEN_FLAG_BITS,
      /*Entry usage flag, '1'B: in use, '0'b: not used*/
    5 BM$TDEN_FLAG_ENTRY_IN_USE              BIT(1),
      /* Deletion protection flag, '1'B:do not delete */
      /*  directory entry without privilege          */
    5 BM$TDEN_FLAG_PROTECTED                 BIT(1),
      /* Data Element entry queue flag               */
      /*  '1'B: entry is part of a queue             */
    5 BM$TDEN_FLAG_QUEUE_MEMBER              BIT(1),
      /* Data Element entry queue header flag        */
      /*  '1'B: entry is the head of a queue         */
    5 BM$TDEN_FLAG_QUEUE_HEAD                BIT(1),
      /* Data Element entry name array flag          */
      /*  '1'B: entry is a name array member         */
    5 BM$TDEN_FLAG_NAME_ARRAY_MEMBER         BIT(1),
      /* Data Element entry name array head flag     */
      /*  '1'B: entry is the head of a name array    */
    5 BM$TDEN_FLAG_NAME_ARRAY_HEAD           BIT(1),
      /* Data Element Type Descriptor Directory      */
      /*  '1'B: entry is a Type Descriptor Directory */
    5 BM$TDEN_FLAG_TYPE_DIR                  BIT(1),
      /* Not yet used bits                           */
    5 BM$TDEN_FLAG_REST                      BIT(25),
      /* OFFSET to Directory extent                  */
3 OM$TDEN_EXTENT_OFF                  OFFSET,
      /* Data Type index of preceding name           */
3 LM$TDEN_PRE_NAME_INDEX              BIN FIXED (31),
      /* Data Type index of following name           */
3 LM$TDEN_FOLLOW_NAME_INDEX           BIN FIXED (31),
      /* binary name tree weight of this entry       */
3 LM$TDEN_NODE_WEIGHT                 BIN FIXED (31),
      /* index of data Area of Data Type             */
3 LM$TDEN_DE_AREA_INDEX               BIN FIXED (31),
      /* OFFSET to data in Area of Data Type         */
3 OM$TDEN_DE_START_DATA               OFFSET,
      /* length of Data Type in bytes                */
3 LM$TDEN_DE_LENGTH                   BIN FIXED (31),
      /* index of Type Descriptor of Data Type       */
3 LM$TDEN_DE_TYPE_INDEX               BIN FIXED (31),
```

```
        /* Data Descriptor structure like VMS          */
    3 SM$TDEN_DE_DESC UNION,
        /* Data Descriptor longword                    */
      4 LM$TDEN_DE_DESC                  BIN FIXED (31),
        /* Data Descriptor parts                       */
      4 SM$TDEN_DE_DESC_PARTS,
        /* Data Descriptor, Length of data item in bytes*/
        /* length in bits for bit,                     */
        /* length of an array element for arrays       */
        5 IM$TDEN_DE_DESC_LENGTH         BIN FIXED (15),
        /* Data Descriptor, data type code             */
        5 HM$TDEN_DE_DESC_DTYPE          BIN FIXED (7),
        /* Data Descriptor, data class code            */
        5 HM$TDEN_DE_DESC_CLASS          BIN FIXED (7),
        /* OFFSET to Data Type Directory Link extent   */
    3 OM$TDEN_LINK_EXTENT_OFF            OFFSET,
        /* index of forward queued Data Type           */
    3 LM$TDEN_FORWARD_QUEUE_INDEX        BIN FIXED (31),
        /* index of backward queued Data Type          */
    3 LM$TDEN_BACKWARD_QUEUE_INDEX       BIN FIXED (31),
        /* index of Data Type name array head          */
    3 LM$TDEN_NAME_ARRAY_HEAD_INDEX      BIN FIXED (31),

        /* start of Directory extents                  */
  2 LM$TDEN_FIRST_EXTENT          BIN FIXED (31);
```

C.10 Type Descriptor

A Type Descriptor is a structure describing a Data Element of a Data Base.

To get the PL/I structure into a procedure include: SM$TYDSC with structures SM$TYDSC, SM$TYLIM, and SM$TYREF which will now be listed:

```
/* GOOINC(SM$TYDSC), Structure for type declarations                  */

%REPLACE name_length BY 31; /* N*4 BYTES                              */

DCL P_sm$tydsc POINTER;
DCL P_sm$tylim POINTER;
DCL P_sm$tyref POINTER;
DCL L_sm$tydsc BIN FIXED (31);/* Used for allocate of all three       */
                              /* structures                           */

%REPLACE sm$tydsc_valid BY '10101010101010101010101010101010'B;
%REPLACE sm$tydsc_no_valid BY '01010101010101010101010101010101'B;
DCL 1 sm$tydsc based(P_sm$tydsc),
      2 B_sm$tydsc_valid      BIT (32) aligned,/* control pattern     */
      2 P_sm$tydsc_u1 UNION,
        3 P_sm$tydsc_next     POINTER,
        3 L_sm$tydsc_next     BIN FIXED (31),
      2 P_sm$tydsc_u3 UNION,
        3 P_sm$tydsc_ref_obj  POINTER, /* pointer to refer object     */
        3 L_sm$tydsc_ref_obj  BIN FIXED (31),
      2 L_sm$tydsc_length     BIN FIXED (31),
      2 H_sm$tydsc_type       BIN FIXED (7),
      2 H_sm$tydsc_class      BIN FIXED (7),
      2 CV_sm$tydsc_name      CHARACTER (name_length) VAR,
      2 P_sm$tydsc_u2 UNION,
        3 P_sm$tydsc_extent   POINTER,      /* pointer to pointerlist */
        3 L_sm$tydsc_extent   BIN FIXED (31),/* index to pointerlist  */
      2 P_sm$tydsc_u4 UNION,
        3 P_sm$tydsc_data     POINTER, /*pointer to data (-1 for refer*/
        3 L_sm$tydsc_data     BIN FIXED (31), /* object               */
      2 H_sm$tydsc_scale      BIN FIXED (7),
      2 H_sm$tydsc_digits     BIN FIXED (7),
      2 B8_sm$tydsc_flags     BIT (8) aligned,
      2 H_sm$tydsc_dimens     BIN FIXED (7),
      2 SA_sm$tydsc_limits    (L_sm$tydsc REFER(H_sm$tydsc_dimens)),
        3 LA_sm$tydsc_lower   BIN FIXED (31),
```

```
        3 P_sm$tydsc_u6 UNION,
          4 PA_sm$tydsc_ref_obj_low  POINTER, /* poin.to refer object */
          4 LA_sm$tydsc_ref_obj_low  BIN FIXED (31),
        3 LA_sm$tydsc_upper   BIN FIXED (31),
        3 P_sm$tydsc_u5 UNION,
          4 PA_sm$tydsc_ref_obj_upp  POINTER, /* poin.to refer object */
          4 LA_sm$tydsc_ref_obj_upp  BIN FIXED (31);

DCL 1 sm$tylim based(P_sm$tylim),
      2 L_sm$tylim_dimens       BIN FIXED (31),
      2 LA_sm$tylim_bounds      (L_sm$tydsc REFER(L_sm$tylim_dimens)),
        3 LA_sm$tylim_lower     BIN FIXED (31),
        3 LA_sm$tylim_upper     BIN FIXED (31);

DCL 1 sm$tyref based(P_sm$tyref),
      2 L_sm$tyref_number       BIN FIXED (31),
      2 LA_sm$tyref_ref         (L_sm$tydsc REFER(L_sm$tyref_number)),
        3 CV_sm$tyref_name      CHARACTER (NAME_LENGTH) VAR,
        3 L_sm$tyref_value      BIN FIXED (31);
```

To get the replace definitions of atomic Data Types in a procedure, include: $TYPREP which will now be listed:

```
/* GOOINC($TYPREP)                                                  */
/* Parameters for atomic data types                                 */
/* The code is RANK(char)-64                                        */

%REPLACE typ__H         BY 8;          /* Byte                     */
%REPLACE typ__I         BY 9;          /* Word                     */
%REPLACE typ__L         BY 12;         /* Longword                 */
%REPLACE typ__R         BY 18;         /* Float(24)                */
%REPLACE typ__D         BY 4;          /* Float(53)                */
%REPLACE typ__C         BY 3;          /* Character                */
%REPLACE typ__V         BY 22;         /* Character var            */
%REPLACE typ__B         BY 2;          /* Bit aligned              */
%REPLACE typ__S         BY 19;         /* Structure                */
%REPLACE typ__O         BY 15;         /* Offset                   */
%REPLACE typ__Y         BY 25;         /* Unknown                  */

%REPLACE typ__class_S   BY 19;         /* Scalar                   */
%REPLACE typ__class_A   BY 1;          /* Array                    */
```

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

Contents