U D A S

Data Structure and Data Access Routine

H.Kroiss

IPP 2/288 Aug. 1987

IPP

MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

8046 GARCHING BEI MÜNCHEN

MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

U D A S
Data Structure and Data Access Routine

H.Kroiss

IPP 2/288 Aug. 1987

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Europäischen Atomgemeinschaft über die Zusammenarbeit auf dem Gebiete der Plasmaphysik durchgeführt.

Introduction

The UDAS data format was designed to have the following properties:

The structure of a data file should, if possible,

- be simple and transparent
- contain all information required for complete interpretation of this dataset
- be flexible to allow fast and simple implementation of extensions and modifications
- avoid pointers, offsets, fixed type specifications, fixed sequences and fixed-length structures whenever possible, since such elements complicate the handling of a dataset and, furthermore, restrict the possiblities of the data acquisition system
- contain few different control block types and not prescribe fixed definitions or interpretations of the control blocks
- be organized in records to allow fast access. Each record should only contain a single type of data in order to facilitate conversion to other computer systems.

It was largely possible to satisfy all of the above requirements with the UDAS data format, mainly by means of the character representation (ASCII) of the control and description blocks. The data format, which is composed of only description and data blocks, thereby became very straightforward and flexible.

The individual structure components are described in detail in following sections.

Note

Only the structure of the data format is described, not its interaction with the data acquisition.

CONTENTS

CHAPTER 1

DEFINITIONS

The terms UNIT and LEVEL are used here as follows:

UNIT in this context denotes a logical unit of related data or program options.
This may be, for example, a data acquisition unit, or else a data reduction program or a complete diagnostic. The representation of a unit is always the same, only the interpretation is different. As UNIT was defined in wholly abstract terms, it is possible to map highly different structures on it.

The term LEVEL denotes the grouping of distinct units into a single higher-order data structure. Each level, in turn, may itself be a unit within a higher level. A level may be visualized as a shell enclosing all data structures within it. This recursive data organization provides the advantages of compatibility of the access software at all levels, as well as independence of the nesting depth.

The whole organization of the dataset is composed of just three different types of data blocks:

A. A Unit Access Block (UAB).
The UAB allows fast access to all information in a data package. It is represented wholly in character format (ASCII) and has a special status by virtue of its function as an index. It is always the first block in a dataset and is the only one to be organized in fixed format. Only its length is variable and is governed by the number of units present.

B. The Level Descriptor Block (LDB) and the Unit Descriptor Blocks (UDB).
Both descriptor blocks (LDB and UDB) are represented in free character format (ASCII) and have exactly the same organization. The only difference is the different definition. While the LDB only contains the description section of a whole level (e.g. of a diagnostic or experiment), the UDB only stores information about a single, relevant UNIT.

C. The data blocks (abbreviated to DB).
These contain the actual data of the corresponding level. At the diagnostic level it is the raw data or reduced data, while at the experiment level it is whole sets of diagnostic data that are meant.

The following primitive data types can be specified:

CHAR	the data are of the CHARACTER type
BYTE	the data are of the BYTE type
INT2	the data are of the INTEGER*2 type
INT4	the data are of the INTEGER*4 type
REAL	the data are of the REAL type

Additionally, the following structured data types are defined:

MOD	this refers to data of a unit
DIAG	this refers to data of a diagnostic
EXP	this refers to data of an experiment

CHAPTER 2

GENERAL STRUCTURE

The UDAS datafile has a tree-like structure. The basic pattern can be represented by the following diagrams:

a) Graphs

```
 ____________________________________________________________________________.
!                                                                             !
!                                   EXP                                       !
!                                                                             !
'____________________________________________________________________________'
                    :                    :                    :
 ____________________   _____________   _____________________________________.
!                    ! !             ! !                                      !
!    DIAG 1          ! !   DIAG 2    ! !               DIAG N                 !
!                    ! !             ! !                                      !
'____________________' '_____________' '_____________________________________'
        :        :           :             :          :          :          :
 .______. .______.  .______.   .______.  .______.  .______.  .______.
 !      !!!      !  !      !   !      !  !      !  !      !  !      !
 ! MOD 1!!! MOD 2!  ! MOD 3!   ! MOD 4!  ! MOD 5!  ! MOD 6!  ! MOD N!
 !      !!!      !  !      !   !      !  !      !  !      !  !      !
 '______''______'   '______'   '______'  '______'  '______'  '______'
```

b) Nested parentheses

```
(EXP(DIAG1(MOD1,MOD2),DIAG2(MOD3),DIAGN(MOD4,MOD5,MOD6,MODN)))
```

c) Level representation

```
EXP
        DIAG1
                MOD1
                MOD2
        DIAG2
                MOD3
        DIAGN
                MOD4
                MOD5
                MOD6
                MODN
```

The above representations are different forms of representing the same structure. The 'Graph' representation clearly illustrates the branching. Trees are normally drawn from top to bottom and the top is generally the root. If an element has no successor, then it is called end element or leaf. An element that is not an end element is always a node.

The root of a tree designated by definition is located on level 1. The highest level of a tree is called its depth or height. A further criterion for a tree is the balance. Trees are set to be balanced when the heights of the tree sections for each node differs by one at most.

The number of branches or connections that have to be traversed to the required node is reached from the root is called the path length of the required node. The root has the path length 1.

For structuring the UDAS datafile uses a tree with a degree greater than 2, a so-called balanced "Multipath tree". The path length depends directly on the level of the required node.

This structure proves to be particularly efficient since in most cases the files are stored on magnetic disks and the addresses of several elements located on the same level can be read in one disk access. Compared with a binary tree with several disk accesses, there can be a considerable saving in latency time.

CHAPTER 3

THE DIAGNOSTIC DATASET

A diagnostic dataset represents the smallest unit of a data package and is composed of a unit access block (UAB), followed by a level descriptor block (LDB) and one to n combinations of unit descriptor blocks (UDB's) with data blocks (DB's). The number n here is governed by the number of UNITS available.

Figure 1 shows the structure of a diagnostic dataset

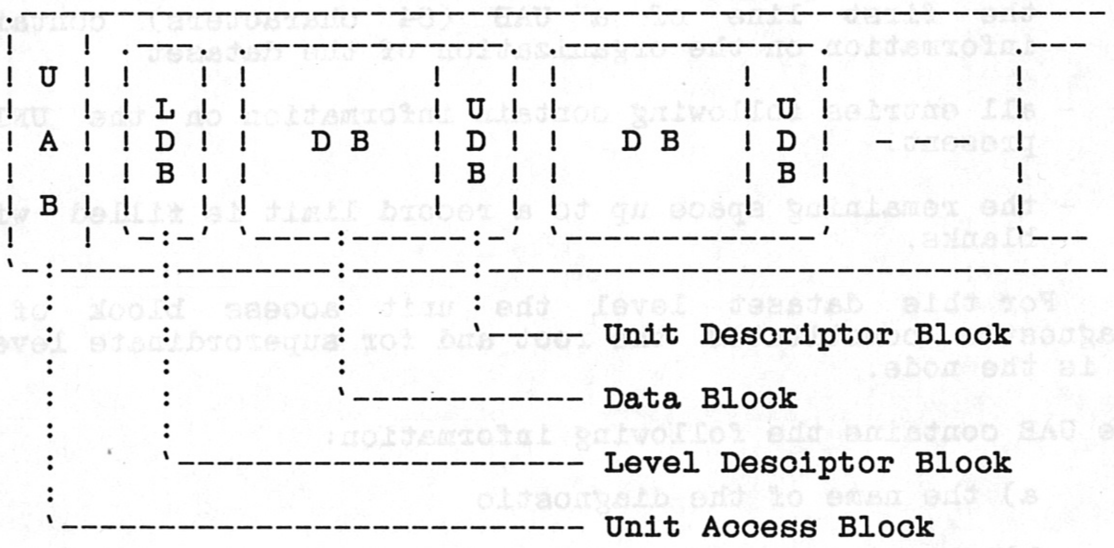


Fig. 1 Organization of a Diagnostic dataset

3.1 Unit Access Block (UAB)

3.1.1 Function of the UAB

The primary function of the unit access block is fast access to data packages within the dataset. Furthermore, it fulfils to a certain extend the function of an index and serves as an information basis for the particular dataset.

For these reasons and in order to ensure the integrity of the datasets for there entire lifetime, the UAB was build in a fixed (and invariable) format. It is thus sufficient just to read out the UABs to identify the particular structure of a dataset. The unit access block is the only element in the data structure to have a rigid format, but the sequence and number of entries has remained variable.

The following peculiarities of UABs should be noted:

- the UAB always stands at the beginning of a dataset
- the format of a UAB entry is fixed (see 3.1.2) and the length is always exactly 64 characters.
- the first line of a UAB (64 characters) contains information on the organization of the dataset
- all entries following contain information on the UNITS present.
- the remaining space up to a record limit is filled with blanks.

For this dataset level the unit access block of a diagnostic constitutes the root and for superordinate levels it is the node.

The UAB contains the following information:

a) the name of the diagnostic

b) the size of a record in bytes

c) the total length of the dataset in bytes

d) the type of the dataset (DIAG)

e) the pointer to the LDB (Level Descriptor Block), i.e. the description section for the DIAG level

f) the length of the LDB in bytes

and for all UNITS involved :

a) the name of the UNIT

b) the pointer to the first data block

c) the number of data points

d) the data type, e.g. REAL

e) the pointer to the UDB (Unit Descriptor Block), i.e. the description section for this UNIT

f) the length of the UDB in bytes

3.1.2 Structure of the UAB

```
                  Unit Access Block    (fixed format)
                  -----------------

 BOLOMETER............. .....512 ....7680 DIAG .......3 .....128
       :                     :         :    :     :        :
       :                     :         :    :     :        :
       `-- Level Name    Rec. Size     :    :     :  Size of LDB
           ( 1-22 )      ( 24-31 )     :    :     :   ( 56-63 )
                                       :    :     :
                                       :    :     : first Record
                                       :    :     `- Number LDB
                                       :    :         ( 47-54 )
                    Total Data  -------'    :
                    ( 33-40 )               `----   Level Type
                                                  ( 42-45 )


 THERMO-ELEMENT........ .......4 ....1280 INT2 .......9 .....343
       :                     :         :    :     :        :
       :                     :         :    :     :        :
       `--- Unit Name  first DB Record :    :     :  Size of UDB
            ( 1-22 )     ( 24-31 )     :    :     :   ( 56-63 )
                                       :    :     :
                                       :    :     : first Record
                                       :    :     `- Number LDB
                                       :    :         ( 47-54 )
            Total Data Items   --------'    :
               ( 33-40 )                    `---- Data Type
                                                  ( 42-45 )


 PHA................... ......10 ....1280 INT2 ......15 .....322
       :                     :         :    :     :        :
       :                     :         :    :     :        :
       `--- Unit Name  first DB Record :    :     :  Size of UDB
            ( 1-22 )     ( 24-31 )     :    :     :   ( 56-63 )
                                       :    :     :
                                       :    :     : first Record
                                       :    :     `- Number LDB
                                       :    :         ( 47-54 )
            Total Data Items   --------'    :
               ( 33-40 )                    `---- Data Type
                                                  ( 42-45 )
```

3.1.3 Example of a UAB

```
---------------------------------------------------------------------
!BOLOMETER                  512      7168 DIAG          3        128
!THERMO-ELEMENT               4      1280 INT2          9        343
!PHA                         10       512 REAL         14        322
!
!
!
!
!
---------------------------------------------------------------------
```

This UAB contains the following information:

This is a diagnostic dataset (DIAG) with the name 'BOLOMETER'. The dataset is organized in 512 byte records and contains a total of 7168 bytes of data. The description section of this diagnostic (LDB) begins with record 3 and contains 128 characters.
In addition, there are two entries for UNITS available. The first unit has the name 'THERMO-ELEMENT', whose data block begins with record 4 and has a length of 1280 data points of type 'INT2', a.e. 2560 bytes. The description section of the 'THERMOELEMENT' unit is found at record 9 and consists of 343 characters.

The second unit 'PHA', can be interpreted by analogy with the unit 'THERMO-ELEMENT'.
The rest of the UAB is filled up with blanks.

3.2 L D B and U D B

3.2.1 The Level Descriptor Block (LDB)

The level descriptor block (LDB) contains all information and parameters concerning the entire level (here: diagnostic). By virtue of this superordinate role the LDB is placed immediatly after the unit access block.
In each LDB the following parameters are to be found:

- the name of the level (here: of the diagnostic)

- the name of the dataset and the shotnumber

- the type of the dataset (here: DIAG)

- the date of generation of the dataset (date, time of day)

Note: Each dataset can only have one UAB and one LDB per level

3.2.2 The Unit Descriptor Block (UDB)

The unit descriptor block contains all information and parameters concerning a special UNIT. This includes essentially parameters which govern the function cycle of the UNIT. The parameters contained in a UDB are grouped in

- a system section, which is governed by the data acquisition program,
- a UNIT section, which reproduces the function cycle of a UNIT (here: unit driver)
- and a user section, which is set up by the user.

The composition and structure of these parameters is of no consequence in this context and is described elsewhere (UDAS Data Acquisition System).

3.2.3 Structure of the LDB and UDB

The structures of the level descriptor block and the unit descriptor block are completely identical. Both are designed in free character format (ASCII) and are unrestricted as regards length, sequence and type of parameters. The following syntactical rules should be observed:

- A LDB or UDB always starts with the name of the level or the unit.
- A parameter name is separated from its value (or its values) by at least a 'blank' or 'TAB'. The same applies to the separation of multiple parameter values.
- A parameter (name and value) is always terminated with a ‹CR› (Carriage return), a.e. a ‹CR› is always followed directly by the name of a parameter or 'blank' (end).

All parameters, including those "invented" by the user, can always be read again under the chosen name for data analysis. By this method it is possible without additional masks or aids to analyse the dataset with the relevant parameters without difficulty from the first to the last shot.

3.2.4 Example of a LDB

Content of a Level Descriptor Block (L D B)

```
Count       Parameter                                Meaning
-------------------------------------------------------------------------
000000      LASER<CR>                              ! Level Name
000006      Data-Set  DN000002.LASER   2<CR>       ! Dataset (Name,Number)
000034      Date-Time  24-MAY-84   12:34<CR>       ! Generation Date
000062      Level-Type  DIAG<CR>                   ! Dataset Type
000080      USER   'Dr. Maier'<CR>                 ! User
000098      System   'VAX11-1'<CR>                 ! Name of Subsystem
000116
   .
   .
   .
000512
-------------------------------------------------------------------------
```

Note: <CR> means "Carriage Return"

3.3 The Data Blocks (DB)

A data block contains all data of a UNIT and has by definition a definitely allocated data type (a.e. REAL). This means that all data of this UNIT, for example, are interpreted as REAL values. A data point thus consists of 4 bytes.

A data block (DB) is always combined with a UDB, which has all the necessary additional information available. In principal, it is possible that the data block is non-existent, i.e. its pointer is zero, while the UDB (at least the UNIT name) always has to be present. This case arises, for example, when a unit, which produces no data requires control parameters.

A data block is physically organized in records which must all have the same data type. This allows the data to be converted relatively simply into other computer representations.
The admissible data types have already been defined under point 1.C. Aside from these data types, user-defined data types can also be selected (e.g. "BCD"). In this case, one byte is taken as the basic unit of data, and the interpretation of the data is left to the user's data analysis program.

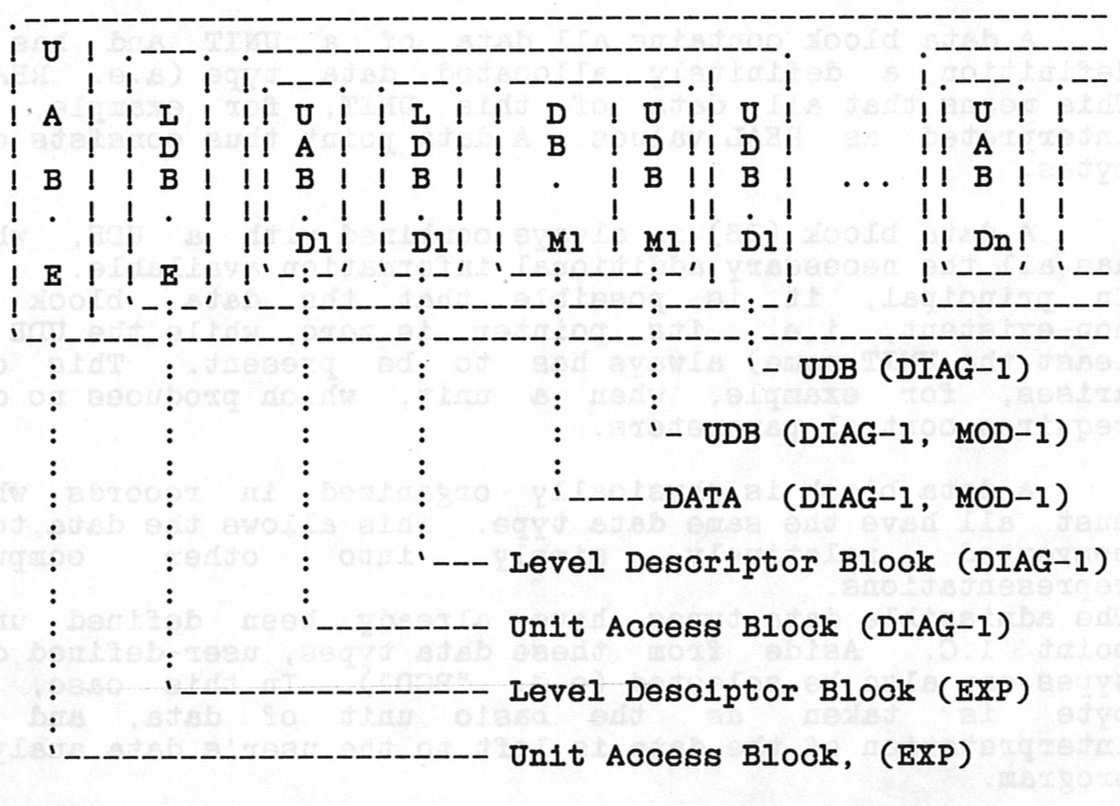
CHAPTER 4

THE EXPERIMENT DATASET

An experiment dataset is composed of the individual diagnostic datasets in form of a shell.

What has been said about the diagnostic datasets (chapter 3) is also basically valid for an experiment dataset. Only the differences in interpretation are therefore discussed in the following.

Figure 2 shows the shell-shaped organization of an experiment dataset.



U A B . E
L D B . E
U A B . D1
L D B . D1
D B . M1
U D B . M1
U D B . D1
...
U A B . Dn
- UDB (DIAG-1)
- UDB (DIAG-1, MOD-1)
---- DATA (DIAG-1, MOD-1)
--- Level Descriptor Block (DIAG-1)
---------- Unit Access Block (DIAG-1)
------------- Level Desciptor Block (EXP)
-------------------- Unit Access Block, (EXP)

Figure 2 Organization of an Experiment Dataset

4.1 The structure of an Experiment Dataset

Each experiment dataset has as "EXP" node a UAB (unit access block), which also represents the root of the entire dataset. All UAB's have a fixed format and afford access to the nodes of the next level. The access scheme is shown in the following representation.

```
 .------------------------------------------------------------------
 !
 !    U  A  B             " E X P "
 !
 !    Unit Access Block    Experiment   "W7AS"
 !
 '------------------------------------------------------------------
    :              :       :       :         :
    v              v       v       v         v
    :              :       :       :         :
 .---------------.         :                 :
 ! L D B  "EXP"  !         :                 :
 !   Level       !         :                 :
 ! Descriptor Block!       :                 :
 '---------------'         :                 :
                           :                 :
 .-----------------------------------------.  .---------------------
 !                                          ! !
 ! U  A  B           " D I A G "            ! ! U A B "D I A G
 !                                          !..!
 ! Unit Access Block  Diagnostik "Bolometer" ! ! Diagnostik  "XY
 !                                          ! !
 '-----------------------------------------'  '---------------------
    :              :    :    :    :        :        :
    v              v    v    v    v        v        v
    :              :    :    :    :        :        :
 .---------------.  :              :
 ! L D B  "DIAG" !  :              :
 !   Level       !  :              :
 ! Descriptor Block! :             :
 '---------------'  :              :
                    :              :
 .------------------------------.   .---------------------------------
 ! U D B          !         !   ! U D B          !
 !        "PHA"   !         !   !         "XYZ"  !
 ! Unit           !  DATA   !...! Unit           !  DATA
 ! Descriptor     !         !   ! Descriptor     !
 ! Block          !         !   ! Block          !
 '------------------------------'   '---------------------------------
```

4.2 Meaning of UAB, LDB and UDB

The unit access block of the experiment dataset has the same structure as a diagnostic UAB. The information, however, is differently interpreted (c.f. see 3.1.1).

The content of the UAB is as follows:

a) name of experiment

b) size of a record in bytes

c) total length of dataset in bytes

d) type of dataset (EXP)

e) pointer to the LDB (Level Descriptor Block), a.e. the description section for the "EXP" level

f) length of the LDB in bytes

and for all UNITS involved (here: diagnostics):

a) UNIT (Diagnostic) name

b) pointer to the UAB of the diagnostic

c) number of data of the diagnostic in bytes

d) data type "DIAG"

e) pointer to the UDB (Unit Descriptor Block), a.e. the description section of the data acquisition for this diagnostic dataset

f) length of the UDB in bytes

Reading this 'EXP' unit access block now affords a quick overview of the structure of the datafile and the diagnostics involved.
The next node (UAB diagnostic), which is the root of a diagnostic dataset, provides detailed information on the particular diagnostic.

The level descriptor block (LDB) of an experiment datasets contains all parameters and information concerning the entire experiment.

On the "EXP" level UNITS are defined as complete diagnostic datasets. The UDB's of an experiment dataset thus contain all parameters and information which are applicable for data acquisition of a diagnostic dataset, a.e. from which subsystem the data came, whether they were subsequnently attached, etc.

The data blocks (DB's) of an experiment dataset are always of the data type "DIAG", a.e. they contain the data of a complete diagnostic, the diagnostic dataset.
Data of type "DIAG" can be interpreted and treated with the structure model of a diagnostic dataset.

CHAPTER 5

DEFINITION OF THE FILENAME FOR DATASETS

All datasets are of the same structure and can be be distinguished by the so-called classification letters. The classification letters always stand at the beginning of a file name.
The classification letters are followed in in file name by a 6-digit number, which corresponds to the respective shot number.
The file extension (VAX) or the file type (IBM) is formed by the first 10 letters of the particular level name.

The name of an UDAS dataset thus consists of three components:

1. two classification letters

2. a 6-digit shot number

3. the level name (1 to 10 characters)

The two classification letters are to allow simple classification of the dataset. The first letter denotes the level of the dataset (a.e. whether it is a unit, diagnostic or experiment dataset):

E --› Experiment

D --› Diagnostic

M --› Unit (module)

The second letter is for allowing pre-analysing of the dataset. The following letters are predefined:

- o N normal rawdata
- o O output data
- o P compressed data
- o R reference data
- o T test data
- o A-M freely selectable qualification

With the qualification letter it is possible, for example, to produce datasets with different data analysis levels. The general structure of a data file name is thus as follows:

```
  L L n n n n n n . L(1)..L(10)
  ! ! !         ! !!           !
  ! ! `---------' !`-----------'
  ! !    !      !      !
  ! !    !      !      `-------- Name of the experiment or
  ! !    !      !                 diagnostic
  ! !    !      `---------------- Separator (only VAX/VMS)
  ! !    `----------------------- 6-digit shot number
  ! `----------------------------- Classification letter
  `------------------------------- Letter for denoting
                                   the level
```

Note: 'L' means letter, 'n' means number

Examples:

EN000001.W7AS	Experiment W7AS, shot number 1 Classification: normal rawdata
DT000002.BOLOMETER	Diagnostic Bolometer, shot number 1 Classification: test data
MK999999.THERMOELEM	Unit Thermoelement, Shot number 999999 Classification: compressed data

The contents of a diagnostic datafile can be added to the experiment datafile with the same shot number. The unit datafiles can likewise be incorporated in diagnostic datafiles (with the same shot number).

The individual diagnostic datafiles can readily be reconstructed (a.e. extracted) from the experiment datafile because the file names are stored in the LDB's of the diagnostic datasets.

CHAPTER 6

FILE ORGANIZATION

The size of the record of a datafile is 512 bytes and corresponds to a physical record of the PDP11/VAX11 file system. The size of the record in the UDAS data structure could vary (see point 3.1.1 "Record Length"), which, however, would certainly lead to complications in many programs. This possibility should only be taken in consideration if the size selected should prove to be unfavourable.

Writing and reading are done by "direct file access". Each datafile record contains only data of the same type (INTEGER*2, REAL, CHARACTER, BYTE...). By means of this condition conversion of the datafile to other computer systems is simplified.

Figure 3 shows the record-wise organization of a datafile

```
.----------                         -------------------.
! 512 Byte      Record 1      CHARACTER    'CHAR'  !
!----------                         -------------------!
! 512 Byte      Record 2      INTERGER*2 'INT2'  !
!----------                         -------------------!
! 512 Byte      Record 3      INTERGER*2 'INT2'  !
!----------                         -------------------!
! 512 Byte      Record 4      INTERGER*4 'INT4'  !
!----------                         -------------------!
!                                                       !
!                                                       !
!                                                       !
!                                                       !
!                                                       !
! 512 Byte      Record n-1  REAL            'REAL'  !
!----------                         -------------------!
! 512 Byte      Record n      CHARACTER    'CHAR'  !
`----------                         -------------------'
```

Figure 3 File Organization

In interpreting the pointers to the unit access blocks (UAB's) its to be noted that the record pointers (which allow direct access) always refer just to their relevant dataset, i.e. the record pointers are to be regarded as relative quantities.

Example:

> In the case of an experiment datafile the start of a diagnostic dataset is indicated in UAB by the record number n. In the diagnostic UAB found as of record number n the number m acts as record pointer to the data of a unit.
> In order to access the data of this unit in 'random access', the record now has to be selected with the absolute number (n+m-1).

This 'relative' organization has proved to be advantageous because the dataset of one level can be transferred untouched (i.e. without recomputing the record pointers) to a dataset on a higher level, and re-extracted later.

CHAPTER 7

UDAS DATA ACCESS ROUTINES

7.1 Overview

The UDAS data files are accessed by means of subroutines (in FORTRAN 77) allowing simple access to the file data and parameters. These routines can be used without modifications to access both diagnostic and experiment data files.

Error handling has been redesigned in relation to previous data access routines. Each function use as result a success/error message in plain text of the data type CHARACTER*80.
The first character of this message indicates if a success, warning or error message is meant. The following specifications are made:

first character = ' '	Success
first character = '-'	Warning
first character = '*'	Error

7.2 Specification and call of functions

To allow the functions be used in the analysis program, they have to be specified as follows:

```
      CHARACTER*80   GETFIL, OPENF, CLOSEF,
     1               SELECT, GETDAT, GETPAR
      CHARACTER*80   RESULT
```

The RESULT variable serves for filing the result of the function.

The function call then reads , for example,

```
      RESULT = CLOSEF ()
```

7.3 Desoription of funotions

7.3.1 GETFIL

A) Desoription:

The GETFIL funotion is only required in VAX/VMS systems in order to affeot synohronization and oommunioation with the 'CP' data aoquisition program.

After data has been written to the file the UDAS data aoquisition system affords the possibility of automatioally starting analysis programs. For this purpose it is neoessary to oommunioate to the analysis program the name of the ourrent shot, whioh oan then be immediately used to open the file.

B) Parameters:

Parameter	Type	Meaning
FNAME	CHARACTER*80 (output)	name of ourrent data file
SHOTNR	INTEGER (output)	ourrent shotnumber

C) Definition of funotion:

```
CHARACTER*80    FUNCTION  GETFIL (FNAME, SHOTNR)
```

D) Remarks:

The oalling of GETFIL is optional (exeption: data reduotion programs). It is reoommended, however, that GETFIL be inoorporated in new programs as a preoaution in order to allow automatio program start - if desired at any later time. Furthermore, it should be noted that, owing to its built-in synohronizing funotion, GETFIL

1. has to be oalled before all other aooess routines
2. and may only be oalled onoe.

E) Error messages:

A WARNING is issued if the analysis program is not automatioally started by the data aoquisition. In this oase the two parameters of GETFIL are irrelevant and the analysis program has itself to define or read in the desired data file name.
This oase always ooours when the user starts the analysis program with 'RUN program'.

F) Example of call:

```
        CHARACTER*80 FNAME
        INTEGER      SHOTNR
        ...
        RESULT = GETFIL (FNAME, SHOTNR)
        IF (RESULT(1:1).EQ.'-') TYPE *,'No automatic mode'
```

7.3.2 OPENF and CLOSEF

A) Description:

The OPENF and CLOSEF routines are available for opening and closing the data file.

To read from a data file, the first essential requirement is to open the file.

Before terminating the program and opening a further data file with OPENF, it is necessary to close the previously opened file with CLOSEF.

B) Parameters:

for OPENF:

Parameter	Type	Meaning
FNAME	CHARACTER*80 (input)	name of data file to be opened
LUNIT	INTEGER (input)	logical unit number is required, to avoid any conflict with other files present in the program

for CLOSEF: none

C) Definition of function:

```
          CHARACTER*80    FUNCTION  OPENF (FNAME, LUNIT)

          CHARACTER*80    FUNCTION  CLOSEF ()
```

D) Remarks:

While a program is running, only one data file may be open at any time !!
When CLOSEF is called, it is no longer necessary to specify the filename and the logical unit number since only one data file can be open.

E) Error messages:

A FATAL error occurs if the data file could not be properly opened. In this case all the other data acess routines cannot be used. (They yield the fatal error 'File not open'). Check to see whether the filename is completely and correctly written and whether the stated data file is indeed present.

If a data file is already opened and has not been closed, OPENF issues a warning and closes the opened data file before the new data file is opened.

F) Example of call:

```
          INTEGER    LUN
...

          LUN = 3
          RESULT = OPENF('DN000001.BOLOMETER', LUN)
          RESULT = CLOSEF()
```

7.3.3 SELECT

A) Description:

In preparation for the GETDAT and GETPAR routines it must be established from which diagnostic or module the data are to be fetched. This is done with the SELECT routine.

The SELECT function must be used before the first call to GETDAT or GETPAR, but can then be used any number of times between OPENF and CLOSEF.

Specifying a diagnostic name and a module name establishes from which diagnostic and module GETDAT and GETPAR derive the data. The selection of the diagnostic and module made by SELECT remains valid for GETDAT and GETPAR calls until diagnostic and/or module is redefined by a new SELECT call.

Access to the experiment parameters is always possible, regardless of which diagnostic and module has just been selected.

B) Parameters:

Parameter !	Type !	Meaning
DIAGN	CHARACTER*22 (input)	name of diagnostic
MODN	CHARACTER*22 (input)	name of module

C) Definition of function:

```
        CHARACTER*80    FUNCTION   SELECT (DIAGN, MODN)
```

D) Remarks:

Between a successfull SELECT call with, for example, a Diagnostic-A and Module-1 and the next SELECT call it is possible to access

- the parameters of the experiment
- the parameters of 'Diagnostic-A'
- the parameters of 'Module-1' and
- the data of 'Module-1'

E) Error messages:

A FATAL error occurs

1. if the data file is not opened
2. if no diagnostic or module name is specified
3. if, in the case of a diagnostic data file, parameter DIAGN does not agree with the name of the diagnostic in the data file.

In the event of a FATAL error all subsequent calls to GETPAR and GETDAT return the error message 'No unit selected'.

A WARNING is issued

1. if, in the case of an experiment data file, the specified diagnostic, and hence the module as well, could not be found.
2. if the specified module could not be found within the diagnostic.

If a WARNING is issued, all following GETPAR calls can access the experiment parameters of an experiment data file and - insofar as the diagnostic could be found - the diagnostic parameters as well.
The data and parameters of a module cannot be read.

F) Example of call:

```
        RESULT = SELECT ('Diagnostic-A', 'Module-1')
```

7.3.4 GETDAT

A) Description:

The GETDAT routine is used for actual data access.

GETDAT may be used between OPENF and CLOSEF any number of times. The first GETDAT call must, however, be preceeded by a SELECT call.

B) Parameters:

required:

Parameter	Type	Meaning
BUFFER	Array (output)	the type of BUFFER must agree with the data type
DSTART	INTEGER (input)	starting location of data
DATREQ	INTEGER (input)	number of data points requested

optional:

Parameter	Type	Meaning
DATEND	INTEGER (output)	end location of data supplied
DATRET	INTEGER (output)	number of datapoints actually supplied
DFORM	CHARACTER*4 (input)	expected format of data ('REAL', 'INT2', 'INT4', 'CHAR', 'BYTE')

C) Definition of function:

```
          CHARACTER*80       FUNCTION  GETDAT
     1              (BUFFER, DSTART, DATREQ, DATEND, DATRET, DFORM)
```

D) Remarks:

The first three parameters are essential for every GETDAT call, whereas the other three are optional. But as soon as one of these optional parameters is specified, all parameters to the left of it have to be included as well, a.e. if GETDAT is called whith a parameter for DATRET, the DATEND parameter may not be omitted.

The DATEND parameter facilitates complete reading of the data by using DATEND+1 as DSTART input for the particular GETDAT call following.

To avoid an abnormal end of program, it should be insured when calling GETDAT

1. that the specified buffer is large enough to take the data requested and
2. that its type agrees with the type of data.

If there are doubts concerning the type of data involved, the optional DFORM parameter should be used. DFORM can take as value one of the usual abbreviations for denoting type of data ('REAL', 'INT2', 'INT4', 'CHAR', 'BYTE', etc.) and sould inform the GETDAT routine what type of data is expected (a.e. how BUFFER is defined). If the data type and buffer type do not agree, an error message is issued, but no data are transfered.

The buffer type must be defined accordingly for the data types REAL, INT4, INT2 and CHAR.
The size of the buffer can be arbitrarily selected.

For example:

```
      REAL              BUFFER(1000)       for REAL data

      INTEGER*4         BUFFER(400)        for INT4 data

      INTEGER*2         BUFFER(256)        for INT2 data

      CHARACTER*1       BUFFER(512)        a.e.
      CHARACTER*512     BUFFER             for CHAR data
```

A datapoint here corresponds to a REAL number, an INTEGER*4 number, an INTEGER*2 number, or a CHARACTER*1.

For all other types of data (BYTE,...) BUFFER has to be defined as an array of LOGICAL*1 values, for example:

```
      LOGICAL*1         BUFFER(400)
```

In this case a datapoint denotes one byte.

E) Error messages:

A FATAL error occurs

1. if fewer than the first three parameters are specified,
2. if the data file has not been opened with OPENF,
3. if there has previously been no proper SELECT call to choose then module from which the data are to be read,
4. if the datatype does not agree with the DFORM parameter.

A WARNING is issued

1. if no data from the module selected are available in the data file,
2. if no data are available at the specified starting location,
3. if more data are requested than are present.

F) Example of call:

```
           INTEGER*2        BUFFER(500)
           INTEGER          DSTART, DATREQ, DATEND, DATRET
           ...
           DSTART = 1
           DATREQ = 500
           RESULT = GETDAT(BUFFER, DSTART, DATREQ,
        1                    DATEND, DATRET, 'INT2')
```

7.3.5 GETPAR

A) Description:

The GETPAR routine allows simple access to the parameters of the experiment, of the diagnostic, or of a module.

Like GETDAT, GETPAR may also be used between OPENF and CLOSEF any number of times after (at least) one SELECT call.

With one GETPAR call it is possible to obtain the values of up to eight parameters all at once.

B) Parameters:

Parameter !	Type !	Meaning
UNIT	CHARACTER*4 (input)	selects what kind of parameter is to be read 'EXP ' for experiment parameter 'DIAG' for diagnostic parameter 'MOD ' for module parameter
PNAMEx	CHARACTER*22 (input)	name of parameter
PBUFx	record (output)	the type of PBUFx is dependend on the type of parameter buffer for the value of the parameter

C) Definition of function:

```
      CHARACTER*80  FUNCTION  GETPAR (UNIT,
     1                                PNAME1, PBUF1,
     1                                PNAME2, PBUF2,
     1                                PNAME3, PBUF3,
     1                                PNAME4, PBUF4,
     1                                PNAME5, PBUF5,
     1                                PNAME6, PBUF6,
     1                                PNAME7, PBUF7,
     1                                PNAME8, PBUF8)
```

D) Remarks:

GETPAR has to be called with an odd number of arguments (at least three).

The specification for UNIT may under no circumstances be omitted. For every parameter required it is first necessary to specify its name and then the appropriate buffer which is to hold the value.
The UNIT argument establishes whether the following parameter names involve experiment, diagnostic, or module parameters:

```
UNIT = 'EXP ' experiment parameter
UNIT = 'DIAG' diagnostic parameter
UNIT = 'MOD ' module parameter
```

PNAMEx must contain the name of the parameter required, beginning with the leftmost character. An unambigous abbreviation of parameter names is allowed, but not recommended (for reasons of efficiency).

PBUFx is a variable of a type corresponding to the type of the parameter value. The following types are possible:

1. REAL
2. INTEGER
3. CHARACTER
4. and ARRAYS of these types.

If a paramater has several values of the CHARACTER type, these are seperated by a BLANK and written in succession into the character buffer provided for the parameter.

If a parameter should have several values of different types, the relevant buffer must first be defined as an array of LOGICAL*1 values. An EQUIVALENCE statement may then be used to define specific offsets as variables of the desired data type. In this case GETPAR first writes all REAL values, then all INTEGER values and finally all CHARACTER values into the buffer (see definition of parameter "Conversion" in example pp. 17-18). Each REAl or INTEGER value requires 4 bytes; each letter of a character string 1 byte of storage space.

Caution: If the types of the parameter values in the buffer do not agree, this can lead to abnormal end of the program !!

E) Error messages:

A FATAL error occurs

1. if an even number of arguments or fewer than three are specified.
2. if UNIT does not have one of the values 'EXP', 'DIAG' or 'MOD'.
3. if there has prevously been no proper SELECT call,
4. if experiment parameters are requested, although the data file concerned is a diagnostic file.

A WARNING is issued if one or more of the named parameters could not be found.

F) Example of call

```
          INTEGER          DATCNT
          REAL             FACT(6)
          ...
          RESULT = GETPAR ('MOD ',
     1                     'Data-Count', DATCNT,
     1                     'Factors',    FACT)
```

Further GETPAR calls are explained in the second example program.

7.4 Examples of programs

7.4.1 Simple application

```
C
C        Simple example of a program for the application
C        of the UDAS data analysis routines
C

         PROGRAM  SIMPLE
C       *****************************************************
C       The SIMPLE program is used for reading the first 1000
C       datapoints for the 'LASER' diagnostic and the 'ADC-Fast'
C       module from the 'EN000001.W7AS' experiment file.
C       In addition, the parameters 'Data-Count', 'Channels',
C       'Gain' and 'Note' are needed for analysis.
C       *****************************************************

C        Specification of the functions and result message
C        ---------------------------------------------------
         CHARACTER*80       OPENF, SELECT, GETPAR, GETDAT,CLOSEF
         CHARACTER*80       RESULT

C        Specifications for the parameters
C        ----------------------------------
C        The 'Channels' und 'Data-Count' parameters are INTEGER
C        values in the Configuration file, and therefore buffer
C        for them have also to be defined as INTEGERS.
C        For the REAL value of the 'Gain' parameter the REAL
C        buffer GAIN is specified, and for the CHARACTER value
C        of 'Note' the CHARACTER*40 buffer NOTE is specified.

         INTEGER            CHAN,DATCNT
         REAL               GAIN
         CHARACTER*40       NOTE

C        Specification of the buffer for GETDAT
C        ----------------------------------------
         INTEGER*2          I2BUF(1000)
```

```
C       ************* Program Start *************

C       open file EN000001.W7AS
C       -----------------------
        RESULT = OPENF('EN000001.W7AS, 3)
        IF (RESULT(1:1) .EQ. '*') STOP

C       select diagnostic 'LASER', module 'ADC-Fast'
C       ---------------------------------------------
        RESULT = SELECT ('LASER', 'ADC-Fast')
        IF (RESULT(1:1) .EQ. '*') STOP

C       get desired parameters from unit 'ADC-Fast':
C       ---------------------------------------------
        RESULT = GETPAR ('MOD ','Data-Count',    DATCNT,
     1                          'Channels',      CHAN,
     1                          'Gain',          GAIN,
     1                          'Note',          NOTE)

        IF (RESULT(1:1) .EQ. '*') STOP

C       get the first 1000 datapoints:
C       ------------------------------
        RESULT = GETDAT (I2BUF, 1, 1000)
        IF (RESULT(1:1) .EQ. '*') STOP

C       Now you can work with data and parameters
C            . . .

C       close file
C       ----------
        RESULT = CLOSEF()

        END
```

7.4.2 Complex application

```
C
C       Sample program for application of the UDAS
C       data evaluating routines
C


        PROGRAM  BSP

C       Specification of the functions and result message
C       ------------------------------------------------
        CHARACTER*80       GETFIL, OPENF, CLOSEF,
     1                     SELECT, GETPAR, GETDAT

        CHARACTER*80       RESULT

C       File name, logical unit number, shotnumber,
C       Diagnostic and module names
C       ------------------------------------------
        CHARACTER*50       FNAME
        INTEGER            LUNIT, SHOTNR
        CHARACTER*22       DIAGN, MODN


C       Specifications for the parameters
C        Factor_1, Factor_4, System and User
C       ----------------------------------
        INTEGER            FACTA1(3), FACTA4(4)
        CHARACTER*10       SYSPAR
        CHARACTER*100      USERB

C       Specifications for the parameters
C        Frequency and Text
C       ---------------------------------

        REAL               FREQU
        CHARACTER*40       TEXT


C       Specifications for the parameter Conversion
C       Conversion         2.442E-3 4095 2048
C       ------------------------------------------
        LOGICAL*1          CONVER(12)
C       Note: for each REAL or INTEGER value
C             4 bytes storage will be needed

        REAL               CONV1
        INTEGER            CONV2, CONV3

C       Superpositioning of the REAL and INTEGER values
C       on the LOGICAL*1 buffer:
C       -----------------------------------------------
        EQUIVALENCE        (CONVER(1),CONV1)
```

```
      EQUIVALENCE     (CONVER(5),CONV2)
      EQUIVALENCE     (CONVER(9),CONV3)

C     Note:
C          CONVER is superposed in such a way that the REAL
C          value can be accessed with CONV1, and the two
C          INTEGER values with CONV2 and CONV3.

C     Specifications of parameter Factor_2 on module ADC-E1:
C     Factor_2        afafaf 12 1.0e16 volt energie H^13 ende
C     -------------------------------------------------------

      LOGICAL*1       FACTE1(36)
      REAL            FREAL
      INTEGER         FINT1, FINT2
      CHARACTER       AF*6, VOLT*4, ENER*7, ENDE*4

C     Note:
C          GETPAR yields the values of Factor_2 in the
C          following order:
C          first the REAL value in bytes 1-4,
C          then the two INTEGER values in bytes 5-8 and
C          9-12 respectively, and finally the 4 CHARACTER
C          values as of byte 13, each separated by a blank.

      EQUIVALENCE     (FACTE1(1), FREAL)
      EQUIVALENCE     (FACTE1(5), FINT1)
      EQUIVALENCE     (FACTE1(9), FINT2)
      EQUIVALENCE     (FACTE1(13),AF)
      EQUIVALENCE     (FACTE1(20),VOLT)
      EQUIVALENCE     (FACTE1(25),ENER)
      EQUIVALENCE     (FACTE1(33),ENDE)


C     Specification of the buffer for GETDAT
C     ---------------------------------------
      INTEGER*2       I2BUF(2560)
      REAL            REABUF(200)
      CHARACTER*80    CHRBUF
      INTEGER         DATRET, DATEND


C     ************* Program Start **************

      LUNIT = 3

C     try to get current shotnumber by the data acquisistion
C     -------------------------------------------------------
      RESULT = GETFIL(FNAME,SHOTNR)
      IF (RESULT(1:1).NE.' ') FNAME = 'DN000001.NEUTRALINJ'
```

```
C       open the diagnostic data file
C       ----------------------------
        RESULT = OPENF(FNAME, LUNIT)
        TYPE*, RESULT


C       select diagnostic Neutralinjection, Module ADC-A1
C       -------------------------------------------------
        DIAGN = 'Neutralinjection'
        MODN = 'ADC-A1'
        RESULT = SELECT (DIAGN, MODN)
        TYPE*, RESULT

C       get diagnostic parameter "System"
C       ---------------------------------------------
        RESULT = GETPAR('DIAG','System',SYSPAR)
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
        PRINT*, 'System: ', SYSPAR

C       get module parameters "Conversion",
C         "Factor_4", and "Factor_1"
C       -------------------------------------------------
        RESULT = GETPAR('MOD ',
     1                  'Conve',CONVER,
     1                  'Factor_4',FACTA4,
     1                  'Factor_1',FACTA1)
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
        PRINT*, 'Conversion: ', CONV1, CONV2, CONV3

C       select module ADC-E1 in the same diagnostic
C       -------------------------------------------------
        RESULT = SELECT('Neutralinjection','ADC-E1')
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT

C       get module parameter "Frequency" of ADC-E1
C       -------------------------------------------------
        RESULT = GETPAR('MOD ','Frequency',FREQU)
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
        PRINT*, 'Frequency: ', FREQU

C       get module parameter "Factor_1" and "Factor_2"
C         of ADC-E1
C       -------------------------------------------------
        RESULT = GETPAR('MOD ',
     1                  'Factor_1',TEXT,
     1                  'Factor_2',FACTE1)
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
        PRINT*, 'Text: ', TEXT
        WRITE(*,'(A11,A6,1X,I2,1X,E10.2,1X,A4,
     1        1X,A7,1X,Z3,1X,A4)')
     2        'Factor-2: ', AF, FINT1, FREAL, VOLT,
     3                      ENER, FINT2, ENDE

C       get data
C       select Neutralinjection, ADC-E1
```

```
C       expected data format INT2 causes an error
C       ------------------------------------------------
        RESULT = GETDAT(I2BUF,DATEND+1,2560,
     1                  DATEND,DATRET,'INT2')
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT

C       select new
C       ------------------------------------------------
        RESULT = SELECT(DIAGN,'ADC-A1')
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT

C       loop to get all data of ADC-A1
C       ------------------------------------------------
        DATEND = 0
        DO I=1,200
          RESULT = GETDAT(I2BUF,DATEND+1,2560,
     1                    DATEND,DATRET,'INT2')
          IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
          IF (RESULT(1:1).NE.' ')  GOTO 200
        END DO
200     CONTINUE

C       get some REAL data of ADC-C1
C       select ADC-C1
C       ------------------------------------------------
        RESULT = SELECT(DIAGN,'ADC-C1')
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT

C       get data form first to 200th data item
C       ------------------------------------------------
        RESULT = GETDAT(REABUF,1,200)
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
        PRINT*, 'Real Data of ADC-C1: '
        PRINT*, REABUF

C       get CHARACTER data of ADC-D1
C       ------------------------------------------------
        CHRBUF = ' '
        RESULT = SELECT(DIAGN,'ADC-D1')
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT

        RESULT = GETDAT(CHRBUF,100,20)
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT
        PRINT*, 'Character Data of ADC-D1: '
        PRINT*, CHRBUF


C       close file
C       ------------------------------------------------
        RESULT = CLOSEF()
        IF (RESULT(1:1).NE.' ')  TYPE*, RESULT

        END
```