DAFT

User's Manual

DAFT - Data Aquisition System for TVC (Transputer based VME Contoller)

15. 11. 94

DEUTSCHES ELEKTRONEN-SYNCHROTRON
Institut für Hochenergiephysik
Platanenallee 6 - 15738 Zeuthen

First Published, October 1994

© Bert Schöneich, Gunter Trowitzsch

The following are trademarks:

OpenVMS and PATHWORKS are trademarks of the Digital Equipment Corporation.

UNIX is a registered trademark of the X/Open Company Limited.

INMOS Limited is a member of the SGS-THOMSON Microelectronics Group

Contents

1 Introduction

2 Starting DAFT

3 User Interface

4 VME Instruction Language

5 Data Output Handling and File Format

Figures

1 Introduction

Preface

This manual introduces DAFT - Data Acquisition for TVC based Systems.

DAFT is a simple Data Acquisition System for test environments. It based on a single VME-crate controlled by the TVC (Transputer based VME Controller) and a PC. The PC is working as the I/O-server.

This manual describes:

- the condition for using DAFT
- how to operate the system
- examples for the data evaluation

More About DAFT

The main parts of the system are the user interface, the event handler (data output generator) and the handler for the VME crate and the VME modules.

Via the user interface the user is able to control the DAQ session by the help of a simple menu including the control of the DAQ run. There are 3 basic operation modes:

- writing data to an output file (File or DAQ Mode)
- dummy read out of the VME crate (Read-only or Scope Mode)
- dumping data on the screen (Screen Dump Mode)

The event handler writes the data into a file or dumps the data on the screen. The ZEBRA sequential I/O format (FZ format) is used for writing the output file.

To handle the VME side the user has to specify two instruction sequences:

- for the initiation of the run
- and for the read-out of the event data.

DAFT writes a protocoll of all executed commands and actions in a log file. The default file name is `DAFT.LOG`.

2

Starting DAFT

This chapter describes:

- The conditions for using DAFT
- The start procedure

Conditions to Use DAFT

The main hardware components are:

- one VME crate with a TVC (Transputer based VME Controller) and one CORBO VME Read-out Control Board.
  For more information about the TVC refer to the TVC-documentation: H. Leich, Transputer based VME Controller - TVC, IfH Zeuthen, March 1994.
  The CORBO module is used to synchronize the DAQ data read-out and any kind of trigger system. The module is documented in: RCB 8047 CORBO VME Read-Out Control Board, User's Manual, CREATIVE ELECTRONIC SYSTEMS S.A., Geneva, Switzerland, August 1993.
- one personal computer as server for the Transputer system.

The required software components are:

- INMOS Transputer development and server system. At least the server (iserver) must be installed to run DAFT. To modify DAFT the C toolset is necessary.
- DAFT itself.

For large applications and a large amount of data it is useful to connect the PC with some kind of file server to hold the data output files. In the IfH-implementation PATHWORKS is used on an OpenVMS system.

Starting DAFT

Before starting DAFT, all related and necessary files has to be created. These are:

- `DAFT.INI`
  containing all startup parameter and file specifications.
- VME/run initiation instruction file
  containing the VME instruction sequence for starting a run.
- VME read-out instruction file
  containing the VME read-out instruction sequence.

The format of the `DAFT.INI` file is fixed and an example is shown in Figure 2-1.

Figure 2-1 `DAFT.INI` **file**

```
/daft.ini, created by SAVE_PARAMETER at 19940919 130249
/rename this file to daft.ini, then use it
/
EVN 32              /number of eventbuffers in ring (1 ... 32)
EVL 16384           /eventlenght - integer (1 ... 16384)
MEV 900000          /max number of events
/
RNN 1               /runnumber
RTY 111             /runtype : 1 = pedestal
                    /          2 = others
RMO 2               / runmode : 1 : to file
                    /           2 : dummy read
                    /           3 : screen dump
/
RSQ read.vme        /readout-sequence-file
VSQ init.vme        /vme-ini-sequence-file
OUT daft_out.tat    /daft-output-file
/end daft.ini
```

Each parameter must be prefixed by a three character key. The meaning and the limits are given in comments.

Chapter 5 describes the format of the VME instruction sequences.

After preparing the initiation file and the VME instruction sequences, you have to start DAFT by the following command:

`iserver /sb daft.btl`

During the startup phase the following steps are executed:

- Initiation of the TVC
- Reading the `DAFT.INI` file
- Opening the log file `DAFT.LOG`
- Compiling the specified VME/run initiation sequence
- Compiling the specified read-out sequence
- Initiation of the CORBO module
- Starting the processes:
  - User interface (daft_01 on low priority)
  - Event handler (event_p on low priority)
  - VME and interrupt handler (ir_hand on high priority)

The CORBO channel 1 is used in DAFT. The BUSY output is set to "not ready" in the startup initiation, because the NIM level output is used. This signal should be used to stop the trigger system of the test environment. Until the start of a run no TRIGGER is accepted.

3

User Interface

Basic Screen

The basic screen lay-out is shown in Figure 3-1. In the upper part of the screen we found the most important parameters of the system. The values of the changeable parameters are highlighted.

The lower part contains the main menu.

Figure 3-1 Basic Screen Lay-out

```
DAFT - data acquisition for transputer          last action: 16:35:49
---------------------------------------------------------------------------------
ini file: daft.ini          number of buffers: 32
-- -----            ---
    RUN             EVENTS             log file     : daft.log
number:     1   max size: 16384        readout Seq: read.vme
type:     111   to take: 900000        VME ini Seq: init.vme
mode: 2 dummy read                     DAQ output   : daft out.tat
---------------------------------------------------------------------------------
    please take a selection:
    ------------------------

     0 : exit
     1 : modify parameter
     2 : save parameter
     3 : compile VME sequences
     4 : START RUN

    Enter choice : _
---------------------------------------------------------------------------------
```

You can select the desired activity by entering the corresponding digit.

Modify Parameter

The command "1" (Modify Parameter) allowes the change of all highlighted parameters. The Figure 3-2 showes the screen lay-out for this command.

Figure 3-2 Modify Parameter Screen

```
DAFT - data acquisition for transputer          last action: 16:35:49
--------------------------------------------------------------------------
ini file: daft.ini          number of buffers: 32
-- RUN ------      EVENTS  ---   flog file     : daft.log
anumber:     1  dmax size: 16384  greadout Seq: read.vme
btype:     111  eto take: 900000  hVME ini Seq: init.vme
cmode: 2 dummy read               iDAQ output : daft_out.tat
--------------------------------------------------------------------------
please take a selection:
    ------------------------

    0 : exit
    1 : modify parameter             Enter key: _
    2 : save parameter                   (0 : exit)
    3 : compile VME sequences
    4 : START RUN

   Enter choice : 1
--------------------------------------------------------------------------
```

In front of each parameter a blinking letter (from a to i) comes up. This letter has to be specified for the selection.

a : The Run number must be in the range from 0 to 99999. In run mode "1" the number is incremented automatically at the end of the run.

b : The Run type may be used to specify different run conditions, data types or trigger conditions. The legal range is from 0 to 99999.

c : The run mode controls the general operation scream of the system. Only mode 1, 2 and 3 are legal.

- 1 : In mode "1" the events will be written into the DAQ output file.
- 2 : In mode "2" the event buffers are released immediately after reading the event data from the VME bus.
- 3 : In mode "3" the event data will be dumped on the screen in hexadecimal format.

d : The maximal size of one event is 16384. The minimal size is 1. This parameter is used to reserve the space for the event buffers. It is recommended to use large enough buffers because there is no protection against overwriting the buffer boundary.

e : This parameter contains the number of events to be taken. The range is from 1 to 999999. After reaching this limit, the run will be paused automatically to allow modifications concerning the run or trigger conditions or to stop the run.

f : The logging file will contain a complete protocol of the session. You can change the file by specifying a new file name. In this case the old file is closed and the new file will be opened.

g : You have to specify a file that contains the instruction sequence to read the event data from VME modules to the event buffer. In the startup phase of DAFT the sequence is translated to an internal format automatically. After changing the file, you have to compile it.

h : The handling of the initialization sequence is similar to the read-out sequence ("g"). This sequence will be executed once during starting the run.

i : This file will contain the DAQ data in run mode "1".

All file specifications for "f" to "i" must be in agreement with the DOS conventions. The length is limited to 30 characters. The characters "$" and "_" are legal.
Examples: `:\DIRDAT\DATA.TAT` or `SUBDIR\RUN012.TAT` .

Save Parameter

Figure 3-3 contains the screen lay-out for this command.

You can safe the actual parameter setting into a text file. The file has the same format like the `DAFT.INI` file. If you want to use the saved settings in the startup phase of DAFT you have to rename the file to `DAFT.INI` . Specify only file name and extension.

Figure 3-3 Save Parameter Screen

```
DAFT - data acquisition for transputer          last action: 16:35:49
------------------------------------------------------------------------------
ini file: daft.ini        number of buffers: 32
 - -   - - - - -          - - -
    RUN          EVENTS            log file     : daft.log

number:      1   max size: 16384   readout Seq: read.vme

type:      111   to take: 900000   VME ini Seq: init.vme

mode: 2 dummy read                 DAQ output   : daft_out.tat
------------------------------------------------------------------------------
    please take a selection:
    ------------------------

     0 : exit
     1 : modify parameter
     2 : save parameter            to file: _
     3 : compile VME sequences
     4 : START RUN

    Enter choice : 2
------------------------------------------------------------------------------
```

Compile VME Sequences

Figure 3-4 shows screen lay-out for this command.

During the compilation path the window mode of the screen is left and the scroll mode is used to dump the compilation protocol on the screen. The protocol contains also error messages if there are errors. In debug mode the internal representation is typed out also. You have to quit the compilation by a key stroke.

After compiling the sequence the system returns to the window mode and displays the Compile VME Sequence menu.

START RUN

After selecting the START RUN command the screen lay-out depends on the selected run mode. In mode "3" the window mode is left and the scroll mode is used to type out the event data. An example is shown in Figure 5-1. You have to quit each output page by a key stroke and you have to decide about continuation or aborting/stopping the run. The system returns to the basic menu after the end of the event dump (run).

For the other modes the screen lay-out is shown in Figure 3-5 and Figure 3-6. Figure 3-5 shows the screen during run in progress. Figure 3-6 shows the pause run screen.

Figure 3-4 Compile Sequences Screen

```
DAFT - data acquisition for transputer         last action: 16:35:49
--------------------------------------------------------------------------
ini file: daft.ini          number of buffers: 32

 -- -----  ---------- ---
    RUN           EVENTS              log file     : daft.log

number:     1   max size: 16384     readout Seq: read.vme

type:     111   to take: 900000     VME ini Seq: init.vme

mode: 2 dummy read                  DAQ output   : daft_out.tat
--------------------------------------------------------------------------
    please take a selection:
    ------------------------         compile VME sequence
                                     --------------------
     0 : exit                         0 : exit without compiling
     1 : modify parameter             1 : readout sequence
     2 : save parameter               2 : init sequence
     3 : compile VME sequences        3 : readout sequence in debug mode
     4 : START RUN                    4 : init sequence in debug mode

    Enter choice : 3                  Enter choice : _
--------------------------------------------------------------------------
```

Run in Progress

In this phase the number of taken events and on the bottom of the screen the messages from the VME handler are typed out. You have the possibility to pause the run by command "0" or to abort/stop the run by command "1".

Figure 3-5 Run in Progress Screen

```
DAFT - data acquisition for transputer         last action: 16:35:49
--------------------------------------------------------------------------
ini file: daft.ini          number of buffers: 32

 -- -----  ---------- ---
    RUN           EVENTS              log file     : daft.log

number:     1   max size: 16384     readout Seq: read.vme

type:     111   to take: 900000     VME ini Seq: init.vme

mode: 2 dummy read                  DAQ output   : daft_out.tat
--------------------------------------------------------------------------
    please take a selection:         control run
    ------------------------         -----------
                                      0 : pause run
     0 : exit                         1 : abort run
     1 : modify parameter
     2 : save parameter               Enter choice :_
     3 : compile VME sequences
     4 : START RUN

    Enter choice : 4                 Events taken: 111001 / 900000
--------------------------------------------------------------------------
  Run Time Message:
```

Pause Status

If the run is paused, you have the possibility to change the number of events to be taken (command "3") and the run type (command "2") . This allows to take some pedestal events first and then to change the conditions for the data events and to continue the run. Is the number of events reached, the run will be paused automatically. With command "0" the run is continued. Without changing the number of events only some events are taken and the run control will pause the run again. Command "1" stops the run.

Figure 3-6 Pause Run Screen

```
DAFT - data acquisition for transputer          last action: 16:35:49
------------------------------------------------------------------------------
ini file: daft.ini          number of buffers: 32

-- ----- RUN     EVENTS  ---     log file      : daft.log

number:     1   max size: 16384  readout Seq: read.vme

type:     111   to take: 900000  VME ini Seq: init.vme

mode: 2 dummy read               DAQ output  : daft_out.tat
------------------------------------------------------------------------------
   please take a selection:         control run
   ------------------------         -----------
                                    0 : continue run
    0 : exit                         1 : abort run
    1 : modify parameter             2 : modify run type
    2 : save parameter               3 : modify # of events to take
    3 : compile VME sequences        Enter choice : _
    4 : START RUN

   Enter choice : 4                 Events taken: 111342 / 900000
------------------------------------------------------------------------------
   Run Time Message:
```

4

VME Instruction Language

The general VME instruction format is:

INS {address} {#data} {&mask}

INS is the three character instruction code. The other parts are optional for some instructions. They must be specified as numerical constants. The general radix for ***address***, ***#data*** and ***&mask*** is 16 (hexadecimal). It is possible to change the default radix by a leading "**d**" for 10 (decimal), "**b**" for 2 (binary) and "**o**" for 8 (octal). A leading "x" for 16 is redundant.

You have to specify one line per instruction. For comments the leading character may be a "!", "*", "/" or ";". The rest of the line is ignored.

Figure 4-1 shows an example VME instruction sequence.

BRS - block read from same address

The BRS instruction reads a block of data from the same VME address to the event buffer. For this instruction you may specify the number of data words in a constant (***#data***) or by using the internal counter C . This counter must by filled before by the RTC instruction.
Format:

- **BRS** ***address #data***
- **BRS** ***address*** **C**

BRI - block read incrementing address

The BRI instruction reads a block of data to the event buffer by incrementing the VME address (plus 2 for the implemented A24D16 mode).
The number of words is given by a constant (***#data***) or is taken from the internal counter **C** . This counter must by filled before by the RTC instruction.
Format:

- **BRI** ***address #data***
- **BRI** ***address*** **C**

RAS - read and store

This instruction reads one 16 bit data word from the VME address to the event buffer. Optional the data word may be masked before storing.
Format:

- **RAS** ***address***
- **RAS** ***address &mask***

RAF - read and forget

This is a read from address without saving any data. The read mode is A24D16.
Format:

- **RAF** ***address***

WTA - write to address

This instruction writes ***#data*** to the VME address.
Format:

- **WTA** ***address #data***

MAS - mask and set

The MAS instruction is a read-modify-write instruction. It reads 16 bit data from the VME address, masks the value, sets new data bits by an OR with ***#data*** and writes the value back to the VME address.
Format:

- **MAS** ***address &mask #data***

TEQ -read and test until equal zero

This instruction may be used to wait for a zero condition. After reading 16-bit data from the address and an optional masking, the value is tested. On equal zero the next instruction will be executed. On non-equal the instruction will be repeated. The time out limit (maximal number of repetitions) must be specified in ***#data***. If the time out limit is reached, an error number is stored in the event header and the execution of the sequence is continued.
Format:

- **TEQ** ***address #data*** ! time out error # 1
- **TEQ** ***address &mask #data*** ! time out error # 2

TNE - read and test until not equal zero

This instruction is similar to TEQ, but the test condition is not zero.
Format:

- **TNE** ***address #data*** ! time out error # 3
- **TNE** ***address &mask #data*** ! time out error # 4

RTC - read to counter

This instruction reads 16 bit data from a VME address to the internal counter C. It is possible to mask the data.
Format:

- **RTC** ***address***
- **RTC** ***address &mask***

SOP - start of data portion

EOP - end of data portion

The commands SOP and EOP are used to create a sub-record structure inside the event data for variable length data portions. The substructure is :

NW IDT data words....

NW contains the number of words following and the value is calculated with the EOP instruction. The EOP instruction marks the end of the data portion. IDT is a data identifier, that has to be specified in the SOP instruction (***#data***).

Format:

- **SOP** ***#data***
- ***EOP***

Figure 4-1 Example for an Instruction Sequence

```
!
!   sequence for tests
!
  ras 700060 &00FF        * read 8 bit data
  ras 700062 &FFFF        * cut it to 16 valid bits
  ras 700064 &00FF
  ras 700066 &000F
  ras 700068 &FFFF
  ras 70006A &FFFF
!
  brs 700064 #d10         * read 10 words from '64
  brs 700066 #d10         * and  10       from '66
  brs 700068 #10          * and  16       from '68
  bri 70006A #d12         * read 12 words starting at '6A, '6C,....
!
```

5

Data Output Handling and File Format

The system is using one process to handle the event data. It runs on the same low priority level like the user interface. The run mode parameter determines the activities for the event handling.

Read-only or Scope Mode

This mod is selected by run mode "2" (dummy read). The event handler releases the event buffers immediately. By this way a maximal speed is reached for the read out of the VME crate to allow measurements with an oscilloscope.

Screen Dump Mode

This mode is selected by run mode "3". The event header and the event data are dumped on the screen. You have to quit each output page by a key stroke and you have to decide about continuation or aborting the dump (run). Figure 5-1 shows an example of a screen dump.

Figure 5-1 Screen Dump

```
   8:      5a5a    5a5a    5a5a    5a5a    5a5a    5a5a    5a5a    5a5a
  16:      a5a5    a5a5    a5a5    a5a5    a5a5    a5a5    a5a5    a5a5
  24:      a5a5    a5a5    ff00    ff00    ff00    ff00    ff00    ff00
  32:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  40:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  48:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  56:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  64:      ff00    ff00      ff      6c      6e      70      72      74
  72:        76      78      7a      7c      7e      80
 Hit  a..to abort dump  or  any key to continue

Header  :   EVN      date     time     error    length
        00000002 19940920 180211  00000000     78
D A T A :
   0:      5a5a    a5a5    5a5a    a5a5    ff00      ff    5a5a    5a5a
   8:      5a5a    5a5a    5a5a    5a5a    5a5a    5a5a    5a5a    5a5a
  16:      a5a5    a5a5    a5a5    a5a5    a5a5    a5a5    a5a5    a5a5
  24:      a5a5    a5a5    ff00    ff00    ff00    ff00    ff00    ff00
  32:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  40:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  48:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  56:      ff00    ff00    ff00    ff00    ff00    ff00    ff00    ff00
  64:      ff00    ff00      ff      6c      6e      70      72      74
  72:        76      78      7a      7c      7e      80
 Hit  a..to abort dump  or  any key to continue
```

The header output contains the event number, the date, the time, the error flag word and the length of the data record.

The date format is: year*10000 + month*100 + day.
The time format is: hour*10000 + minute*100 + seconds

The event data are dumped in hexadecimal format.

File or DAQ Mode

This mode is selected by run mode "1" (write to file). The event handler copies the header information and the event data to the output data structure and writes the data to the specified file. After copying the information the event buffer is released.

Data structure and file format are in agreement with the ZEBRA FZ format for Sequential Input/Output.

Output data structure

The internal data structure contains the following parts (in C notation):

```
struct {int mpr[8], mlr[2], mpili[17], mbk[10], mdata[16384];} data_1;
```

mpr contains the FZ physical record control words, **mlr** contains the logical record control words, **mpili** contains the pilot information including the user header vector, **mbk** contains the bank system words (all integer, bank ID = DATA, number of data words) and **mdata** contains the event data.

The user header consists of 6 words in the following order:

1 run number

2 event number

3 data

4 time

5 run type

6 error flag word

There is an one to one mapping between the Transputer word and the FZ 32-bit word structure. Because the byte ordering is vice versa to the FZ exchange format, it is necessary to transform the words in an intermediate step. This will not be done on the Transputer level to save time.

File format

The internal data structure is written in physical records containing 900 words to the output file. Depending on the length of the event data one logical FZ record may consists of several physical records.

Converting the Output to FZ eXchange Format

The first implementation of the converter utility NTOX was done on an OpenVMS system because the server PC of the TVC is using the PATHWORKS file service on that system.

NTOX is simple to use. You have to start it and to specify the file names of the data files. The default extension for the native Transputer output file is .TAT and .FZX for the FZ file in exchange format. If the name of the exchange formatted file is omitted, the name of the Transputer file is used and the extension .FZX is added.

The generated file is a binary direct access file. This must be respected in the evaluation phase. The following section contains two example programs for reading the FZX file.

Example:

```
$ run ntox
_ T800 data File: fz_data.tat
_ FZ eXchange data File:
 FZ data file : fz_data.FZX
                in eXchange direct access format created !
              - Output file (... .FZX) closed !
           4    FZ records/blocks written
```

Examples for Reading the FZ File

After the format conversion the file may be processed on a platform of choice. Because the files are direct access files, you have to open the file in the proper way or to use the C-style input mode for the ZEBRA function calls.

Example 1

```
      PROGRAM FZ_TN

      IMPLICIT NONE

      CHARACTER*60 FILENAME
      INTEGER LQ(40000),IQ(40000),ISTORE(40000),IXSTOR,IXDIV,L
      REAL     Q(40000)
      INTEGER IFENCE(100),LINKS(100),LINKR(100),IUHEAD(100)
      INTEGER IQUEST(100),JBIAS,IUHL,I

      COMMON /REC/ IFENCE,LINKS,LINKR,ISTORE

      EQUIVALENCE(LINKS(9),LQ(9),IQ(1),Q(1)),(LQ(1),L)

      COMMON /QUEST/ IQUEST

      TYPE '(1X,A$)','_ .FZX File: '
      ACCEPT '(Q,A)',I,FILENAME

      IXDIV   = 0                   ! division
      JBIAS   = 2                   ! stand-alone d/s

      CALL MZEBRA(0)

      CALL MZSTOR(IXSTOR,'/REC/',' ',IFENCE,LINKS(1),LINKR(1),
     .            ISTORE(1),ISTORE(20000),ISTORE(40000))

      OPEN (1, FILE=FILENAME,STATUS='OLD',FORM='UNFORMATTED',
     .      ACCESS='DIRECT',RECL=900,BLOCKSIZE=3600)

      CALL FZFILE(1, 900, 'D')      ! 'D' for direct access eXchange format
   10 CONTINUE
      IUHL  = 100                   ! max header length
      CALL FZIN (1, IXDIV, L, JBIAS,' ',IUHL,IUHEAD)

      IF (IQUEST(1).NE.0) GOTO 20

       TYPE ('(x,a,i4,/,4x,a,6i,(/,7X,6i))'),
     .    'Header :       length =',IUHL,
     .    'dec', (IUHEAD(I),I=1,IUHL)
       TYPE ('(4x,a,6z12,/,(7X,6z12,/))'),
     .    'hex', (IUHEAD(I),I=1,IUHL)

       TYPE ('(x,a,i6,10x,a,/,4x,a,6i,(/,7X,6i))'),
     .    'Data    :       length =',IQ(L-1),
     .    'dumping the first 12 data words',
     .    'dec', (IQ(L+I),I=1,12)
       TYPE ('(4x,a,6z12,(/,7X,6z12))'),
     .    'hex', (IQ(L+I),I=1,12)

       CALL MZWIPE(0)               ! release the buffer

      GOTO 10
   20 CONTINUE
      END
```

Example 2

```
      PROGRAM MAIN
*
*
      PARAMETER      (L3CORQ=500000)
      COMMON /GCBANK/ Q(L3CORQ)
*
* *** Notify ZEBRA and HBOOK storage limits to the system
*
      CALL GZEBRA (L3CORQ)
      CALL GZIN
*
*
      END

      SUBROUTINE GZIN
*
      PARAMETER (KWBANK=69000)
      COMMON/GCBANK/NZEBRA,GVERSN,ZVERSN,IXSTOR,IXDIV,IXCONS,FENDQ(16)
     +              ,LMAIN,LR1,WS(KWBANK)
      DIMENSION IQ(2),Q(2),LQ(8000),IWS(2)
      EQUIVALENCE (Q(1),IQ(1),LQ(9)),(LQ(1),LMAIN),(IWS(1),WS(1))
      COMMON/QUEST/IQUEST(100)

      PARAMETER (IHMAX=100)
      DIMENSION IHEAD(IHMAX)
      CHARACTER*80 FILENAME
      INTEGER BLSIZE
*
* *** open the file and initialize FZ (using the C interface)
*
      FILENAME = 'fz_data.dat'
      LUNIN = 1
      BLSIZE = 900
      CALL CFOPEN (IQUEST(1),0,BLSIZE,'R',0,FILENAME,ISTAT)
      IF (ISTAT .NE. 0)  GO TO 91
      CALL FZFILE (LUNIN,BLSIZE,'L')
*
   10 CONTINUE
      NHEAD = IHMAX
      CALL FZIN (LUNIN,0,LMAIN,2,' ',NHEAD,IHEAD)
      IF (IQUEST(1) .NE. 0)  GO TO 92
*
      CALL DZSHOW ('test printout',IXSTOR,LMAIN,'BLV',0,0,0,0)
*
      GO  TO  10
*
   91 CONTINUE
      PRINT *,' CFOPEN: open error'
      RETURN
*
   92 CONTINUE
      IF (IQUEST(1) .GT. 0)  PRINT *,' FZIN: EOF read'
      IF (IQUEST(1) .LT. 0)
     &  PRINT *,' FZIN: bad structure, IQUEST(1)=',IQUEST(1)
      RETURN
*
      END
```