ATL-LARG-99-002
HEC-Note-069
January 1999

Endcap Hadronic Calorimeter Offline Testbeam Software

The hec_adc Package Version 3.6

M. Lefebvre D. O'Neil

Department of Physics and Astronomy
University of Victoria
Victoria, British Columbia, Canada

March 8, 1999

Abstract

Prototypes and first modules of the ATLAS Hadronic Endcap liquid argon calorimeter were subjected to beams in several run periods since 1996. The hec_adc software package allows easy access to data recorded during these run periods, for offline analysis and for online monitoring. This note describes the functionalities of the hec_adc software and how to use it.

Contents

1 Introduction **2**

2 Overview of the hec_adc Package **2**

3 Coding Conventions **4**

4 Installation and Execution **4**
4.1 Installing for the First Time 7
4.2 Upgrading from a Previous Version 8

5 Testbeam Data **8**

6 System Datacards **9**

7 User Files **12**

8 Include Files **13**

9 System Packages **14**
9.1 The Pedestal Package 14
9.2 The Cubic Fit Package 15
9.3 The Digital Filtering Package 17
9.4 The Correlation Package 19
9.5 The Beam Chamber Reconstruction Package 19
9.6 The Calibration Package 20
9.7 The Signal Search Package 21
9.8 The System Histogram Package 22
9.9 The Standard Ntuple Package 23

10 The Online Package **26**
10.1 How to Use the Online Package 26

11 Utilities **28**
11.1 Fortran Utilities 28
11.2 UNIX Utilities 28
11.2.1 multi_run.job 28

12 Example **30**

13 Conclusion **30**

1 Introduction

Prototypes and first modules of the ATLAS Hadronic Endcap liquid argon calorimeter were subjected to beams in several run periods since 1996. The hec_adc software package allows easy access to data recorded during these run periods and has been used for offline analysis and for online monitoring. In addition to providing access to raw and pre-processed data for offline analysis, hec_adc can output a set of standard ntuples which can be readily compared to Monte Carlo and should be easily merged with ntuples from other subdetectors in combined tests.

The first version of the hec_adc package (Version 1.0) was presented at the HEC meeting of 12/07/96. This note refers to Version 3.6 of the hec_adc package. It describes the functionalities of the hec_adc software and how to use it. After a brief overview, the coding conventions adopted for this package are presented. Then section 4 describes the installation and execution of the package. Section 5 explains how to obtain the testbeam data in a form suitable for processesing by the hec_adc package. Section 6 contains a short description of available system datacards. User files are explained in section 7, followed by a listing of the available include files (common blocks) in section 8. Section 9 describes the system packages currently available. Section 10 describes the online mode of hec_adc. Finally, the system utilities and a user example are given in Sections 11 and 12.

The software code and documentation are available in a tarred, gzipped version in the afs directory

`/afs/cern.ch/atlas/testbeam/HEC/offline/dev/ .`

They are also available from the hec_adc homepage located at

`http://wwwhep.phys.uvic.ca/~uvatlas/hec_adc/hec_adc.html .`

The authors would like to thank the members of the Endcap Hadronic and Forward calorimeter community, in particular Matt Dobbs, Andrei Kiryunin, Peter Loch, Andrei Minaenko, Denis Salihagic, Pierre Savard, Dieter Striegel, and Hans-Peter Wellisch for their support and contributions.

Questions or comments about the hec_adc package can be directed to either of `lefebvre@uvic.ca`, `dugan@uvic.ca`, or `uvatlas@pp.phys.uvic.ca`.

2 Overview of the hec_adc Package

The hec_adc package is designed as a simple and hopefully clear framework to allow access to the HEC testbeam data. The following wish list of goals inspired the development of the hec_adc package:

- easy access to the HEC testbeam data;
- independent code development by users;
- easy implementation of matured development code;
- easy maintenance.

The HEC testbeam data contains a limited number of relatively simple data structures. Therefore the hec_adc package was designed with a minimum effort in mind. The following choices have been made:

- FORTRAN language;

- use of include files for common blocks;
- use of the `gmake` facility;
- use of `tar` and `gzip` for distribution.

The hec_adc package does not require the user to know anything about CMZ, ZEBRA or EPIO (apart from the production of the relevant ZEBRA files from the raw EPIO data for 1996 data only). A minimum knowledge of the `gmake` facility is required.

The hec_adc package follows a simple, structured design. The logical structure of the code can be followed starting from file `/src/hec_adc.f` which is the main steering routine. The hec_adc package can be used in two modes:

- The offline mode;
  this mode is the normal mode of hec_adc, and allows offline treatement of the data, and is the subject of this note, unless otherwise specified;
- The online mode;
  this mode is specially tailored to allow hec_adc to be used for online monitoring, and was extensively used for the first time during the April 1998 beam test period. This mode is documented in section 10.

The general flow of the offline mode hec_adc package is shown in Figure 2. Note that each pass over the data is handled by different routines. There is no pass counting variable (`ipass`) in the hec_adc package. We note the following general steps:

- Initialization;
- Execution of datacard driven system packages.

  In this version we have

  - the pedestal package;
  - the cubic fit package;
  - the digital filtering package;
  - the correlation package;
  - the beam chamber reconstruction package;
  - the calibration package;
  - the signal search package;
  - the system histogram package;
  - the standard ntuple package.

  New system packages can initially be developed by users in the form of user routines (see section 7). When mature, they can be promoted by the authors of hec_adc to the level of system packages. These packages will then be installed in hec_adc for all to use. Each system package can require one or more passes over the data set.
- User pass 1 over data (if requested);
- User pass 2 over data (if requested);
- User pass 3 over data (if requested);

- Termination.

Figure 2 also shows the entry points for each user routine. Note that the user must only provide user routines and, possibly, user include files. The program flow for a pass over the data set is shown in Figure 2. The entry points of the pass user routines are also shown. A few relevent include files (see section 8 for the complete list) available to the user are shown in relation to the routine that fills them.

3 Coding Conventions

The hec_adc package is written in FORTRAN. The version chosen for this package is FORTRAN 77 with the following allowed extensions:

- `do while` statements;
- `enddo` statements;
- `include` statements;
- long words.

Further, several conventions have been used in order to make it a clearly written, easy to maintain package. These conventions are as follows:

- many clear and correct comments;
- the use of `implicit none`;
- `print` before `stop`;
- no automatic `save`;
- no `goto`;
- use `include` command for all common blocks;
- commons in documented files, one common per file;
- one routine per file;
- no compilation warnings on `atlas.wgs` machines.

4 Installation and Execution

The installation instructions provided in this section assume the installation is being done on a UNIX workstation. hec_adc should work on other platforms (windows, mac, etc.), assuming the presence of a fortran compiler and the CERN libraries, however, this has never been tested.

The latest public version of the hec_adc package can be found in the directory

`/afs/cern.ch/atlas/testbeam/HEC/offline/dev/`

or from the hec_adc homepage at

`http://wwwhep.phys.uvic.ca/~uvatlas/hec_adc/hec_adc.html` .

Please read the files hec_adc.readme and hec_adc.versions for an update on the version changes and warnings.

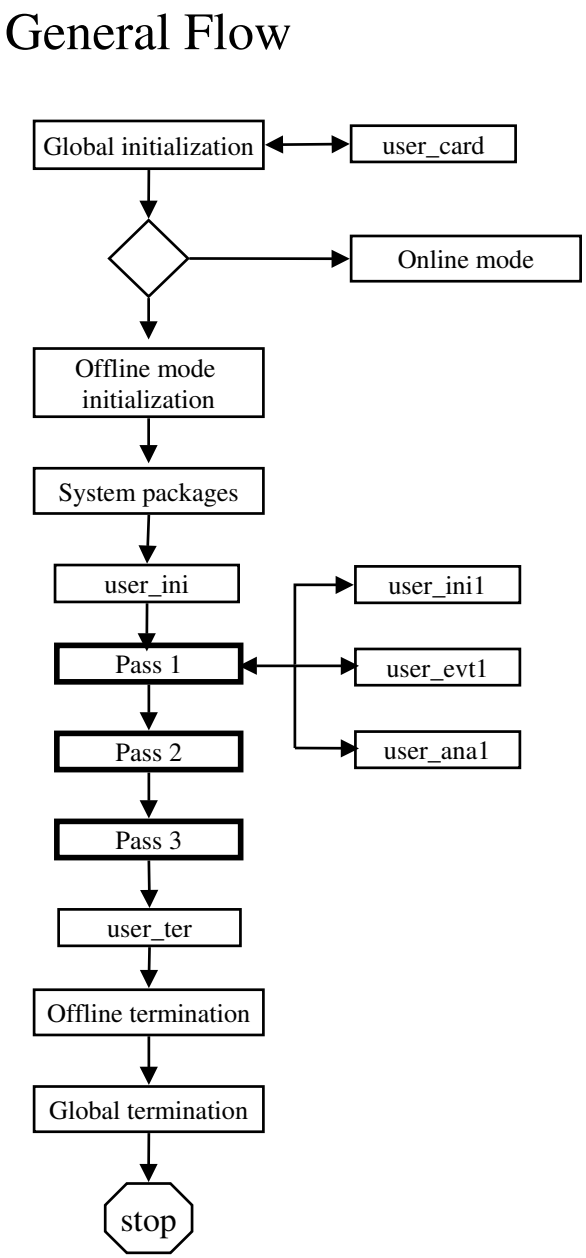

Figure 1: General flow of the hec_adc package. The entry points for the user routines are also shown.

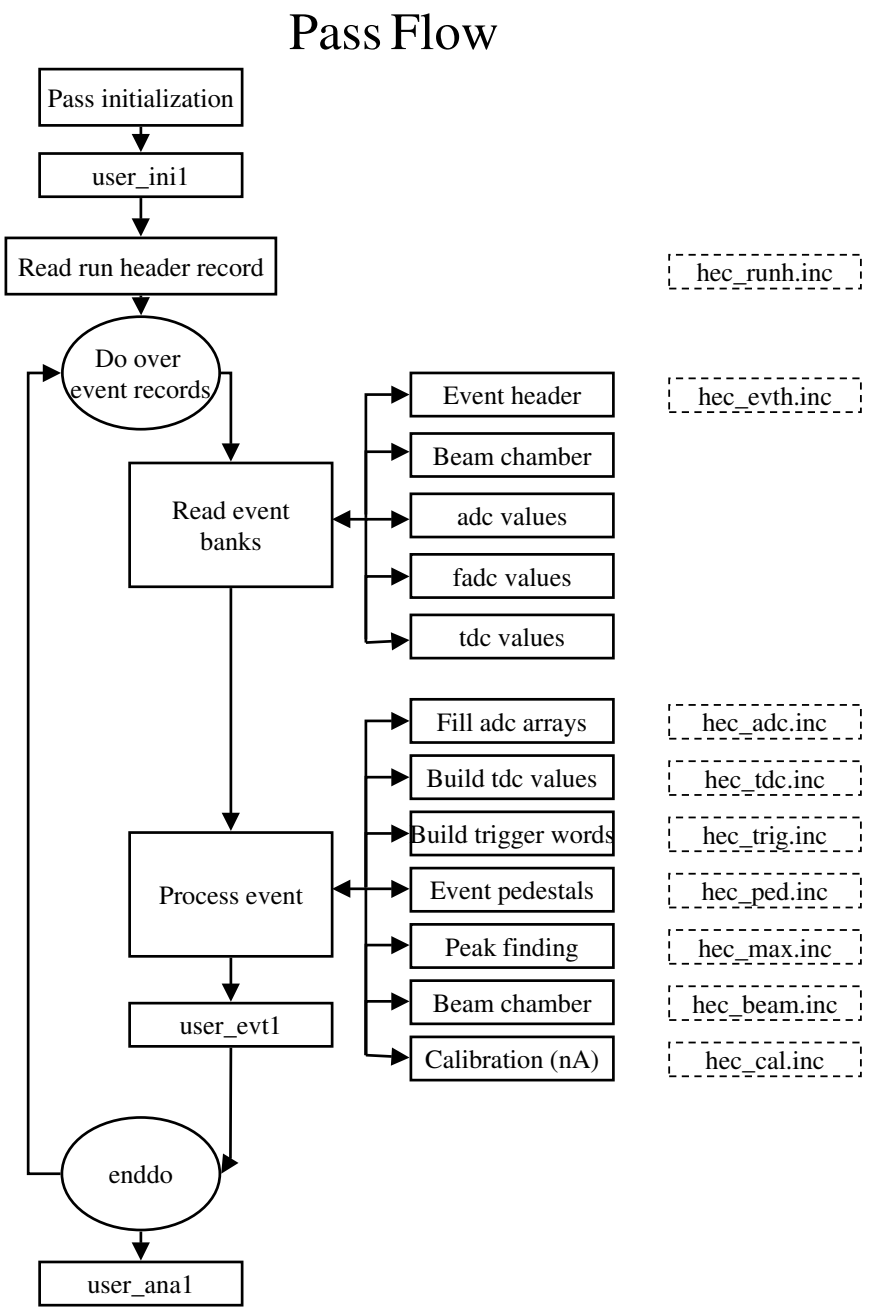

Figure 2: Program flow for pass1 over the data set. Note the location of the pass 1 user routines. Relevant include files are shown in relation to the routine that fills them. The optional pass 2 and pass 3 over the data set follow the same program flow.

4.1 Installing for the First Time

In order to install the hec_adc package, it is necessary to copy the entire directory tree into your working directory. This can be done by putting the file hec_adc.tar.gz into your working directory and typing

```
gunzip  hec_adc.tar.gz
tar xvf hec_adc.tar
```

The `hec_adc.a` library can then be built by entering the hec_adc directory and issuing the command

```
hec_adc_lib.ins
```

This command uses the file `hec_adc_lib.make` and all the fortran and include files in the following directory trees

```
/src     hec_adc fortran source files;
/inc     hec_adc include files containing common blocks;
/util    hec_adc utilities.
```

The resulting object files are neatly kept in the directory

```
/obj     hec_adc object files;
```

Once the library has been built, a sample executable (`hec_adc`) can be produced from the `hec_adc/user` directory by issuing the command

```
hec_adc.ins
```

This command produces the executable `hec_adc`. For its execution, two files are needed: the datacard file and the data file. The datacard file controls the operation of the software with several switches. This datacard file must be named hec_adc.datacard. The data file may be in EPIO (for all data) or ZEBRA (for 1996 data only) format and must be named

```
epioin
```

if it is an EPIO file, or

```
bank.fz
```

if it is an ZEBRA file. The production of such a file from a testbeam data tape is discussed in section 5.

To execute the program using the sample user datacard file provided in the user directory, type

```
hec_adc >myfile.log
```

The output from this test run may include three files:

```
myfile.log        text log file;
hec_adc.hbook     system histograms;
ped_out.dat       pedestals;
ntuple.hbook      standard ntuple output file.
```

In the course of code development and data analysis by the user, only files located in the `hec_adc/user` directory should be modified. A discussion of how a user can modify these routines is included in section 7.

When mature, user routines can be included in the next release of the hec_adc package by the people maintaining it. These routines will be included in the form of an available system package, thereby making this development available to other users.

4.2 Upgrading from a Previous Version

An effort has been made to ensure that new versions of the hec_adc package remain backward compatible with previous versions back to version 2.5. Therefore, to install and run the latest version of the package, it should not be necessary to rewrite user analysis routines. Simply install the latest version in a new directory (as described in the previous section), copy the pre-existing user fortran and include files into the user directory of the new package and recompile. The only file which requires particular care is is the datacard file (`hec_adc.datacard`). It is necessary to change the datacard file from one version to the next in order to incorporate new system packages and to modify existing packages for new data. Therefore, it is advisable to use the most recent version of the datacard file and to copy any user datacards previously created into the new datacard file.

5 Testbeam Data

The hec_adc package can read both raw EPIO data files and ZEBRA datafiles. In order to run on an EPIO data file stored on a CERN tape, enter the HEC staging area:

```
/afs/cern.ch/atlas/testbeam/HEC/data/
```

and issue the command

```
hec_stage firstrun lastrun
```

to stage the data on tape (note: this command will stage all runs between `firstrun` and `lastrun`, see the hec_stage README file in the staging area for details). Then link to the staged file and run the hec_adc package. The name of the EPIO input file should be `epioin`.

The remainder of this section treats the ZEBRA data format, relevant only for 1996 data. To run from a ZEBRA file requires that the input data be in ZEBRA format and be contained in a file named bank.fz. A ZEBRA file for a particular testbeam run can be produced in a two step process. First, stage the tape as described above. Next, once the tape has been staged, the ZEBRA file can be produced with the adc96 program (many thanks to Denis Salihagic for producing it from the old code). In the directory

```
/afs/cern.ch/atlas/testbeam/HEC/offline/dev/zebra/
```

compiled executables of adc96 for May, July and Sept 1996 data can be found. To run adc96 on the ATLAS wgs machines, it is only necessary to take one of these precompiled versions and the file job96.link in order to produce the ZEBRA file. The user can copy the appropriate executable and job96.link to their working area and edit job96.link so that it links to the staged run number. Then type

```
job96.link
```

followed by

```
adc96_myrunperiod.exe > myfile.log
```

and a ZEBRA file named bank.fz will be produced. This file will be the input to the hec_adc package. Three example zebra files have also been included in this directory.

If you wish to recompile adc96, the files needed are also included in

```
/afs/cern.ch/atlas/testbeam/HEC/offline/dev/zebra/
```

In this case, the user should copy the files adc96.car and job.ins into their working directory and type

```
job.ins
```

Several files will be created, including adc96.e, which is an executable version of adc96 (the run period is chosen within job.ins). Also, a CMZ file will be produced containing the contents of adc96.car (this might be easier to edit than the .car file). Note that job.ins will always compile the card file, so any changes made to the cmz file must be exported to a card file in order to take effect.

Once the new adc96 executable has been created, use it as explained above. As of yet, no version of adc96 exists for data after 1996.

6 System Datacards

The user can control various options of the hec_adc package with datacards. Their values are set in the file `/user/hec_adc.datacard`. The following system datacards are available:

```
rnmode  i  1= offline mode
           2= online mode (overrides some other datacards)

intyp   i  1= input data file type flag (1 is ZEBRA)
                                       (2 is EPIO)
        i  2= logical unit for data file reading

npass   i  1= number of analysis passes (<4)

nevtm   i  1= maximum number of events to analyse per pass

runpd   i  1= 1 may96 run period
           1= 2 jul96 run period
           1= 3 sep96 run period
           1= 4 feb97 run period (technical)
           1= 5 may97 run period
           1= 6 oct97 run period
           1= 7 apr98 run period
           1= 8 aug98 run period

mkped   i  1= 1 make peds for each event only
           1= 2 make peds over all events only
           1= 3 both methods, but prefer for each event
           1= 4 both methods, but prefer over all events
           1<=0 do not make peds
        i  2= first time sampling (meaningful if irunpd>3)
        i  3= last time sampling  (meaningful if irunpd>3)
        r  4= sigma cut for peds production (for over all events)
              if <= 0, then no cut and only one pass is used
        i  5= 1 use random trigger events only (automatic if irunpd<4)
           5= 2 use pretrigger
           5<=0 use all triggers
        i  6= read pedestal file flag    (>0 = read ped)
```

```
        i  7= logical unit for pedestal file reading
        i  8= output pedestal file flag  (>0 = write ped)
        i  9= logical unit for pedestal file writing
        i 10= print flag (=0 do not print the list of pedestals)

mkcub   i  1= find adc max using simple cubic fit (>0 = enable)
        r  2= sigma cut above which the cubic fit is used (<0 = disable)
        i  3= time slice number used if cubic fit not used
        i  4= restrict peak finding time range flag (>0 = enable)
        i  5= min time slice for restricted peak search (if activated)
        i  6= max time slice for restricted peak search (if activated)

mkdig   i  1= find adc max using digital filtering (>0 = enable)

mpref   i  1= prefered adc max finding package
              if more than one adc max finding package
              is enabled, this card determines the
              package that will be used to fill the
              generic adc max common (see hec_max.inc)
           1= 1  for simple cubic fit
           1= 2  for digital filtering
           1= 3  digital filtering but in cells with
                 bad weights use maximum time slice
                 (dig_tzero + 3)
           1= 4  digital filtering but in cells with
                 bad weights use cubic fit

mkhis   i  1= system histograms flag     (>0 = make his)
        i  2= logical unit for system histogram file writing
        i  3= do/don't (1/0) use search algorithm to determine
              which cells will fill histogram

hiscon  i  1= do/don't (1/0) produce pedestal and noise histograms
        i  2= do/don't (1/0) produce geometrical energy distribution histograms
        i  3= do/don't (1/0) produce uncorrected adc histograms
        i  4= do/don't (1/0) produce tdc corrected histograms
        i  5= do/don't (1/0) produce calibrated histograms
        i  6= do/don't (1/0) produce cluster histograms

hisctit##
           Cluster title card. ## is the number of the cluster
           the title refers to (eg. hisctit01 is cluster 1)
        c  1= 4 character title of cluster (eg. 'pi,h')

hisclus##
           Cluster definition card. ## is the number of the cluster
           (eg. hisclus02 is cluster 2)
        i  1= number of cells in cluster (number to follow in list)
        i  2= channel number of first cell in cluster
        i  3= channel number of second cell in cluster
```

```
        i  etc. up to number of cells specified in first argument

iocal   i  1= read calibration file flag (>0 = read cal)
        i  2= logical unit for calibration file reading
        i  3= do/don't (1/0) print calibration constants to screen

rrunh   i  1= correct run number (if <0, ignore)
        r  2= correct beam energy(if <0, ignore)
        i  3= correct beam particle type(if <0, ignore)
        r  4= correct x-position (mm) of cryostat
              (if >10000, ignore)
        r  5= correct y-position (mm) of MWPC moveable table
              (if >10000, ignore)

beamc   i  1= beam chamber reconstruction on/off (1/0)
        r  2= z-coordinate (cm) (along beam), at which lateral
              (x,y)-coordinates of beam should be calculated
        i  3= beam histograms print flag (print, if .ne.0
              and if mkhis flag is on)
        i  4= logical unit for alignment constants reading
        i  5= logical unit for alignment constants writing

mkcrl   i  1= turn channel/time correlation on (>0 = make correlations)
        i  2= logical unit number for crl histo output

search  i  1= search algorithms:<0= search package disabled
                                0= dummy search
                                1= standard search (D. Striegel)
                                2= (0) could be replaced by user
                                   subroutine hec_search_user
           The following values are for the standard algorithm
        i  2= timeslice to start searching
        i  3= timeslice to stop searching
        i  4= number of slices to compare (3, 5, 7)
        l  5= apply a cut on maximum
        r  6= level of the cut

mkntp   i  1= output standard ntuple (1=yes,0=no)
        i  2= logical unit number of ntuple file
        i  3= units of energy in event ntuple
              (1=adc,2=nA,3=GeV)
        r  4= difference from expected time-zero (in ns).
              used to screen out events with bad timing
              (0. deactivates)
        i  5= turn on/off (1/0) ntuple user routines


online  i  1= activate script-controlled online mode (1 = yes)
        i  2= run number fed from UNIX script
        i  3= mon_break
```

The user can also program user datacards, see section 7 for details.

7 User Files

All user routines can be found in the directory

```
hec_adc/user
```

The user can supply the following routines:

```
user_card.f              setting up of user datacards;
user_ini.f               user initialization for run;
user_ini1.f              user pass 1 initialization routine;
user_evt1.f              user pass 1 event routine (called once/event);
user_ana1.f              user pass 1 end of run routine (called once/run);
user_ini2.f              user pass 2 initialization routine;
user_evt2.f              user pass 2 event routine;
user_ana2.f              user pass 2 end of run routine;
user_ini3.f              user pass 3 initialization routine;
user_evt3.f              user pass 3 event routine;
user_ana3.f              user pass 3 end of run routine;
user_ter.f               user termination routine;
user_search.f            user routine for search package
                         (see search documentation);
user_ntuple_block_ini.f  user routine for ntuple package (booking)
                         (see ntuple documentation);
user_ntuple_block_evt1.f user routine for ntuple package (filling)
                         (see ntuple documentation).
```

A dummy version of each of these routines is in the `hec_adc.a` library, therefore only the ones altered by the user need to be supplied in the `hec_adc/user` directory.

During intialization, the hec_adc package will call the `user_card.f` routine, where the user can program user datacards. An example on how to program user datacards can be found in the `user_card.f` file provided. After actually reading the datacards from `hec_adc.datacard`, the `user_ini.f` routine will be called. There, the user can perform global user initializations, like opening an hbook file and booking histograms (note: hbook initializations are already done by the system, you need only open an hbook file, fill it and close it). The values of the system and user datacards are then available and can be used (see section 8).

For the first pass over the data set, the user can provide `user_ini1.f` as the user pass1 inititialization, and `user_evt1.f` as the user routine called for each event, and `user_ana1.f` as the user routine called at the end of the data set. Similarly, if more than one pass is needed, the user can provide `_ini`, `_evt` and `_ana` files for a second and third pass. During termination, the hec_adc package calls the user termination routine `user_ter.f`.

The user is, of course, free to have as many user include files as required to communicate between the user routines. They should be included in the `hec_adc.make` file dependencies, along with any new user routines.

To produce the hec_adc executable, the user needs to execute the file

```
hec_adc.ins
```

This command uses the file `hec_adc.make`, the library `hec_adc.a` and all the user*.f files. The resulting user object files remain on the `hec_adc/user` directory. It is up to the user to

maintain both `hec_adc.ins`, where the links to other files are set, and `hec_adc.make`, the make file instructing how to build the executable.

For each user file to compile, the `hec_adc.make` file needs only to contain reference to user include files (ie. any .inc files created by the user).

The user also needs to supply the file `hec_adc.datacard` that contains the user's choice of datacard values. System and user datacards must be put in `hec_adc.datacard`. The program can then be executed from the directory `hec_adc/user` by typing, for example,

```
hec_adc > myfile.log
```

8 Include Files

The user has access to the data through include files located in the directory `hec_adc/inc`. Hopefully, they contain enough comments to be self documented. **Please read them carefully**. The only include file which is required by other include files is hec_par.inc. This include file contains the array dimensions used in many other include files and therefore should be present at the top of most routines (it must precede the other include files).

System include files can be grouped as follows:

- include files relevant to all user*.f routines:

```
hec_par.inc         fixed parameters, also used by other include files;
hec_datacard.inc    variables associated with datacards.
```

- include files relevant after initialization

```
hec_cal_coef.inc    calibration coefficients (if read from file);
hec_geo.inc         geometry correspondence tables and bad cells list;
hec_runh.inc        run header info (RUNH bank).
```

- include files relevant for each event of a run

```
hec_adc.inc         adc values;
hec_beam.inc        beam particle trajectory;
hec_cal.inc         calibrated energy (nA);
hec_crl.inc         information from correlation package;
hec_cub.inc         signal maximum information from cubic fit package;
hec_dig.inc         signal maximum information from digital filtering
                    package;
hec_evth.inc        event number and pattern units (EVTH bank);
hec_fadc.inc        flash adc values;
hec_max.inc         pedestal subtracted peak height and time for multiple
                    time sample data filled by ''preferred'' method;
hec_ntuple.inc      ntuple variables;
hec_online.inc      online mode information;
hec_ped.inc         adc pedestal and pedestal rms values;
hec_search.inc      info from signal finding algorithm;
hec_tdc.inc         tdc values;
hec_trig.inc        trigger words.
```

- include files containing raw data or used by the system (generally of no interest to the user):

```
hec_beam_sys*.inc    used by beam chamber package;
hec_dig_sys.inc      used by digital filtering package;
hec_dwpc.inc         beam chamber data;
hec_epio.inc         epio common;
hec_his_sys.inc      used by system histogram package;
hec_maxepio.inc      epio buffer size;
hec_ntuple_sys.inc   used by system ntuple package;
hec_paw.inc          paw common;
hec_ped_sys.inc      used by pedestal package;
hec_search_1_sys.inc information for signal search algorithm 1
hec_stats_epio.inc   epio banks statistics
hec_zebra            zebra common.
```

9 System Packages

As mentioned earlier, mature user routines can be made available to other users in the form of system packages. The system packages currently available are described below.

9.1 The Pedestal Package

The pedestal package allows the user to calculate pedestals and their rms for each cell. These quantities can be computed in two general ways in this package:

1- event by event;

2- over all events in a run.

The event by event pedestal calculation is only possible for data taken from February 1997 onwards (run periods 4 and up, data taken with multiple time sampling). It uses a selectable range of time slices, usually the first few. The pedestals and rms are obtained from the mean and rms of the content of these time slices. In particular, if only one time slice is selected, no rms can be computed. The pedestal rms value obtained with more than one time slice, for each event, is in principle independent of the number of time slices used (as long as they are away from the signal time slices). Since these pedestals and rms values are different for each event, they cannot be read from or output to file.

The pedestals and their rms over all events in a run are obtained from the mean and rms of the event by event pedestals calculated over a pass over the data. A second pass may be required if a sigma cut is requested. A summary of the results is printed after each pass. For data taken from February 1997 onwards, if more than one time slice is requested, say n, then the pedestal rms values will approximately scale like $1/\sqrt{n}$. The type of trigger can also be selected. Random triggers are automatically selected for data taken in 1996 (run periods 1 to 3). Pedestals and rms computed over all events can be read from or output to file. Although the pedestal file format is compatible with the old HEC code, it is recommended to produce the pedestals directly in the code. It takes a few seconds on an hp735. Note that the pedestal rms obtained over all events, for run periods 4 and up, represent the fluctuation of the event by event pedestals, each of which are the mean over a selectable time slice window. If the user wishes to produce a pedestal rms indicative of the fluctuation of individual time slices, simply set the time slice range to only one time slice (with $2 = 1$ and $3 = 1$, see below).

Both methods can be requested simultaneously; in this case, the preferred method must be specified. Otherwise, the preferred method is the one selected. The preferred arrays are filled for every event. The values are available to the user via the include file `hec_ped.inc` (which requires the file `hec_par.inc`) which contains the following arrays:

```
adc_ped(ic)     adc pedestal     (real)  Prefered method
adc_rms(ic)     adc pedestal rms (real)
adc_ped_run(ic) adc pedestal     (real)  Computed over a run
adc_rms_run(ic) adc pedestal rms (real)
adc_ped_evt(ic) adc pedestal     (real)  Computed at each event
adc_rms_evt(ic) adc pedestal rms (real)
```

The pedestal package is steered by the following datacards:

```
mkped  i  1= 1 make peds for each event only
          1= 2 make peds over all events only
          1= 3 both methods, but prefer for each event
          1= 4 both methods, but prefer over all events
          1<=0 do not make peds
       i  2= first time sampling (meaningful if irunpd>3)
       i  3= last time sampling  (meaningful if irunpd>3)
       r  4= sigma cut for peds production (for over all events)
       i  5= 1 use random trigger events only (automatic if irunpd<4)
          5= 2 use pretrigger
          5<=0 use all triggers
          (5 is only for over all events)
       i  6= read pedestal file flag    (>0 = read ped)
       i  7= logical unit for pedestal file reading
       i  8= output pedestal file flag  (>0 = write ped)
       i  9= logical unit for pedestal file writing
       i 10= print flag (=0 do not print the list of pedestals)
```

For example,

```
mkped  1=3  2=1  3=4  4=1.5  5=0  6=0  7=48  8=1  9=49  10=1
```

instructs the pedestal package to first compute pedestals and rms over all events using two passes over the data set, with a 1.5 sigma cut on the second pass, and to output the result to file. Then, pedestals are also computed at every event during the user passes. The user can then access, for each event, pedestals and rms computed using both methods: event by event in arrays `adc_ped_evt` and `adc_rms_evt`, and over all events in arrays `adc_ped_run` and `adc_rms_run`. Since 1 = 3 was selected in this example, the preferred arrays `adc_ped` and `adc_rms` are filled, every event, with the pedestals and rms computed for that event. These preferred arrays allow the user to write code that can accept any types of pedestals, selected using the datacard. The pedestal list will be printed.

9.2 The Cubic Fit Package

Communication with the cubic fit package is done using the following datacard:

```
mkcub   i  1= find adc max using simple cubic fit (>0 = enable)
        r  2= sigma cut above which the cubic fit is used (<0 = disable)
        i  3= time slice number used if cubic fit not used
```

```
i  4= restrict peak finding time range flag (>0 = enable)
i  5= min time slice for restricted peak search (if activated)
i  6= max time slice for restricted peak search (if activated)
```

The purpose of the cubic fit package is to find the signal peak height (in adc counts) and time (in time slice units) for an individual channel in an individual event. The cubic fit method provides a quick, simple method to fit the signal peak and can act as a cross-check of more sophisticated fitting methods such as digital filtering.

The output of the package is contained in the include file `hec_cub.inc` and consists of the array `adc_cub_ic` with the following structure:

```
adc_cub_ic(ic, 1)  max adc value in channel ic
adc_cub_ic(ic, 2)  max adc t value in channel ic in units of time samplings
```

The time slice with the highest adc signal becomes the search seed. The time slice window in which the seed can reside can be set with the parameters 2, 3 and 4 of the datacard. The seed time slice becomes the second or third time slice of a fit window of 4 time slices. The cubic fit package then fits a cubic function to the 4 time samplings in the fit window. The cubic function is:

$$Y(t') = a_0 + a_1 t' + a_2 t'^2 + a_3 t'^3$$

where the a_i are 4 fit parameters and $t' = t - t_0$, and t_0 is the first time sampling used in the fit. This provides a system of four equations with 4 unknowns which can easily be solved to obtain the four coefficients of the fit. The time of the maximum (in units of time samplings) is then obtained from the expression:

$$t'_{\max} = \frac{-a_2 - \sqrt{a_2^2 - 3a_1 a_3}}{3a_3}$$

and the peak height is then obtained from the cubic equation above with $t' = t'_{max}$. For events in which the method fails, the peak is set to the seed time slice signal and time.

This fitting procedure works well for channels containing a high signal, however, it has been found to produce biased results for low signal channels. In order to reduce this bias, two datacard switches (2 and 3 above) have been provided. The first switch sets the sigma threshold above which the cubic fit is a valid fitting method. For example, if this value is set to 5, then any channel containing a fitted maximum adc more than 5 sigma above the pedestal will be considered a good cubic fit; for any channel below this threshold an alternate maximum will be used. The simplest alternate maximum is used, this is the adc value at a certain timeslice as set with datacard switch 3. Therefore, if this method is activated the cubic fit will be used for high signals and the value of the chosen timeslice will be used for low signals.

Since this software contains a choice of multiple signal fitting packages, a generic set arrays can be filled with the peak height and time from the fitting package preferred by the user. The fitting package used to fill these arrays is chosen with the datacard

```
mpref  i  1= preferred adc max finding package
             if more than one adc max finding package
             is enabled, this card determines the
             package that will be used to fill the
             generic adc max common (see hec_adc_max.inc)
          1= 1  for simple cubic fit
          1= 2  for digital filtering
```

```
1= 3  digital filtering but in cells with
      bad weights use maximum time slice
      (dig_tzero + 3)
1= 4  digital filtering but in cells with
      bad weights use cubic fit
```

For example `m_pref 1=1` fills the generic array with the output of the cubic fit package, while `m_pref 1=2` will fill these same arrays with the output from a different fitting method. These general arrays are in the include file `hec_max.inc` and have the structure (**all max adc values are pedestal subtracted**):

```
adc_max_ic(ic, 1)                    max adc values   "adc channel system"
adc_max_ic(ic, 2)                    max adc t values in units of ns

adc_max_db(i_mod, i_seg, i_pad, 1)   max adc values   "database system"
adc_max_db(i_mod, i_seg, i_pad, 2)   max adc t values in units of ns

adc_max_ph(i_eta, i_phi, i_z, 1)     max adc values   "physics system"
adc_max_ph(i_eta, i_phi, i_z, 2)     max adc t values in units of ns
```

9.3 The Digital Filtering Package

Communication with the digital filtering package is done using the following datacard:

```
mkdig   i  1= find adc max using digital filtering (>0 = enable)
```

The purpose of the digital filtering package is to find the signal peak height (in adc counts) and time (in ns) for an individual channel in an individual event. This method of determining the time and height of maximum signal should provide the best possible results in terms of noise performance while maintaining good calculation speed.

The digital filtering method, as implemented in the hec_adc code, requires two files of parameters, one set for amplitude reconstruction, and one set for time reconstruction. These parameters are used to calculate the digital filtering weights for each channel, for each event. For example, for April 1998 data, the time dependence of the weights is parametrized with a polynomial of 4th order. So, there are 5 parameters needed to calculate each individual weight. The layout of the files of parameters is as follows:

```
version number (time stamp eg. 981014)
number of slices used in filtering (eg. 5)
number of constants used to calculate weights for one channel (eg. 5)
first time slice used in peak reconstruction (eg. 5)
channel  weight |             File Content
-----------------------------------------------------
   1       1    |  par1    par2    par3    par4   par5
   1       2    |  par1    par2    par3    par4   par5
   1       3    |  par1    par2    par3    par4   par5
   1       4    |  par1    par2    par3    par4   par5
   1       5    |  par1    par2    par3    par4   par5
   2       1    |  par1    par2    par3    par4   par5
   2       2    |  par1    par2    par3    par4   par5
   2       3    |  par1    par2    par3    par4   par5
   2       4    |  par1    par2    par3    par4   par5
```

```
2       5     |  par1    par2    par3    par4    par5
.       .     |   .       .       .       .       .
.       .     |   .       .       .       .       .
.       .     |   .       .       .       .       .
```

The formula for obtaining the ith weight for the kth channel is then

$$w_i^k = \sum_{l=0}^{4} \mathrm{par}_i^k (l+1) \cdot t_{\mathrm{TDC}}^l.$$

Using this expression weights can be calculated for amplitide (w_a) and time (w_t) reconstruction. Once these weights are known, the amplitude in a particular channel can be calculated using

$$\mathrm{Amplitude} = \sum_{i=1}^{5} (w_a)_i \cdot S_{5+i-1}$$

where S_i is the pedestal subtracted adc value of the ith time sampling. After the reconstruction of the amplitude, calibration coefficients can be applied to convert pedestal subtracted adc to nA. Similarly, time can be reconstructed using

$$\mathrm{Amplitude} \cdot \tau = \sum_{i=1}^{5} (w_t)_i \cdot S_{5+i-1}$$

where τ is the time difference (in ns) from the expected t_0.

The output of the package is contained in the include file `hec_dig.inc` and consists of the array `adc_dig_ic` with the following structure:

```
adc_dig_ic(ic, 1)  max adc value in channel ic
adc_dig_ic(ic, 2)  max adc t value in channel ic in units of ns
```

Since this release of this software contains a choice of multiple signal fitting packages, a generic set arrays can be filled with the peak height and time from the fitting package preferred by the user. The fitting package used to fill these arrays is chosen with the datacard

```
mpref  i  1= preferred adc max finding package
             if more than one adc max finding package
             is enabled, this card determines the
             package that will be used to fill the
             generic adc max common (see hec_adc_max.inc)
          1= 1  for simple cubic fit
          1= 2  for digital filtering
          1= 3  digital filtering but in cells with
                bad weights use maximum time slice
                (dig_tzero + 3)
          1= 4  digital filtering but in cells with
                bad weights use cubic fit
```

These general arrays are in the include file `hec_max.inc` and have the structure (**all max adc values are pedestal subtracted**):

```
adc_max_ic(ic, 1)                  max adc values   "adc channel system"
adc_max_ic(ic, 2)                  max adc t values in units of ns
```

```
adc_max_db(i_mod, i_seg, i_pad, 1)   max adc values   "database system"
adc_max_db(i_mod, i_seg, i_pad, 2)   max adc t values in units ns

adc_max_ph(i_eta, i_phi, i_z, 1)     max adc values   "physics system"
adc_max_ph(i_eta, i_phi, i_z, 2)     max adc t values in units of ns
```

9.4 The Correlation Package

Communication with this package is done with the following datacard:

```
mkcrl   i  1= turn channel/time correlation on
              (>0 = make correlations)
        i  2= logical unit number for crl histo output
```

The correlation package processes random trigger events and its purpose is to calculate:

- for each adc channel and for each time sampling the average signal and r.m.s. (pedestal and noise)
- for each adc channel the correlation coefficients between different time samplings
- for each time sampling the correlation coefficients between different adc channels

The output of the package is contained in 3 ntuples in the an hbook file `crl.hbook`. Ntuple 1 provides pedestal and noise information, ntuple 2 provides time sampling correlations and ntuple 3 provides channel-to-channel correlations. Utilities (kumacs) have been provided in the `hec_adc/util/crl` directory for processing these ntuples.

For further information on this package contact Andrei Kiryunin (Andrei.Kiryunin@cern.ch).

9.5 The Beam Chamber Reconstruction Package

Communication with the beam chamber package is done with the following datacard:

```
beamc   i  1= beam chamber reconstruction on/off (1/0)
        r  2= z-coordinate (cm) (along beam), at which lateral
              (x,y)-coordinates of beam should be calculated
        i  3= beam histograms print flag (print, if .ne. 0
              and if mkhis flag is on)
        i  4= logical unit for alignment constants reading
        i  5= logical unit for alignment constants writing
```

The beam chamber package is designed to use beam chamber information to reconstruct particle direction (and hence can be used to calculate the impact point on the calorimeter).

The package fills a common block in the `hec_beam.inc` include file with the following variables:

```
xbeam - beam x-coordinate at z=zbeam
ybeam - beam y-coordinate at z=zbeam
zbeam - z-coordinate, at which (x,y)-coordinates of a beam trajectory
        are calculated. It should be set during the initialization stage.
xslop - trajectory x-slop: at any z     x=xbeam+xslop*(z-zbeam)
yslop - trajectory y-slop: at any z     y=ybeam+yslop*(z-zbeam)
xbeam_ok - .TRUE., if beam x-projection is successfully reconstructed
ybeam_ok - .TRUE., if beam y-projection is successfully reconstructed
```

This package requires knowledge of the alignment of the beam chambers in order to fill the beam common block. The package attempts to read the required alignment parameters from a file called `align.dat` in the directory in which hec_adc is being run. If this file is not present the package passes over the data to calculate the required parameters and creates the `align.dat` file.

For further information about this package contact Andrei Minaenko (Andrei.Minaenko@cern.ch).

9.6 The Calibration Package

Communication with the calibration package is done through the following datacard:

```
iocal  i  1= read calibration file flag (>0 = read cal)
       i  2= logical unit for calibration file reading
       i  3= do/don't (1/0) print calibration constants to screen
```

The calibration package is designed to convert the raw adc values obtained from the EPIO or ZEBRA input into energies measured in nano-amps. It reads a user provided calibration file that must be named `calib.dat`. A link to an example of such a file for will be automatically produced by `hec_adc.ins`. The file will be formatted as in the following example

```
981014
    1   0.00000e+00  2.95729e+01 -1.47677e-06  2.18080e-11   2
    2   1.80468e+02  3.52081e+01 -2.06677e-06  2.41705e-11   2
    3   1.03668e+01  3.23534e+01 -3.80024e-06  5.02904e-11   2
    4  -1.35757e+02  3.59394e+01 -4.30791e-06  5.65738e-11   2
    5  -1.40582e+01  3.02743e+01 -7.09976e-07  1.09726e-11   1
    .       .            .            .            .         .
    .       .            .            .            .         .
    .       .            .            .            .         .
```

where the first line contains the version number (date stamp) of the calibration file. The calibration constants start on the second line; the first column of numbers is the channel number, the next four columns are four calibration coefficients and the final column is a status number (>2 is a dead channel, 1 is a good channel). The calibration package will read this file and apply a calibration according to the following formula for runs taken from October 1997 onwards:

$$\text{energy} = c_2(\text{ADC}) + c_3(\text{ADC})^2 + c_4(\text{ADC})^3.$$

Please note that the first calibration constant (c1) is now ignored for energy reconstruction. For previous periods the formula applied is:

$$\text{energy} = c_1 + c_2(\text{ADC}) + c_3(c_1 + c_2(\text{ADC}))^2 + c_4(c_1 + c_2(\text{ADC}))^3$$

where the energy calculated is the energy for a particular channel, c_1, c_2, c_3 and c_4 are the 4 calibration coefficients read in from the calibration file and ADC is the pedestal subtracted adc content of that channel.

Using this information, the calibration package fills the `hec_cal.inc` include file with arrays of the energy in each cell in each of the three coordinate systems. The three arrays are as follows:

```
hec_cal_ic(ic)              the energy in cell ic
hec_cal_db(imod,iseg,ipad)  the energy in module=imod, segment=iseg and pad=ipad
hec_cal_ph(ieta,iphi,iz)    the energy in eta=ieta, phi=iphi, z=iz
```

The user should access these arrays in order to use calibrated energies.

9.7 The Signal Search Package

Signal Search Package (by D. Striegel)

The signal search package looks for cells with a valid signal shape in time according to a certain algorithm (hence the package is valid only for data taken from February 1997 onwards, that is run periods 4 and up).

The results of the signal search for each event is available in the file `hec_search.inc`.

```
hec_s_chan(ic)
   .false. = no valid signal found in channel ic
   .true.  = valid signal found in channel ic
   if no algorithm selected, all .true.

hec_s_result(ic, j)
   for a channel with valid signal, adc value of channel ic
   where j is a time slice index such that adc max is at j=3
   for a channel with no valid signal, all zeros.
   meaningful only if hec_s_chan(ic) is .true.

hec_s_im(ic)
   index of timeslice (i_t) of the adc max which is at j=3
   in hec_s_result
   meaningful only if hec_s_chan(ic) is .true., but not
   meaningful for the dummy search algorithm (number 0)

hec_s_hit
   .true. = at least one channel with valid signal
   if no algorithm, then .true.

hec_s_cells
   number of channels with valid signal
   if no algorithm, this is equal to i_adc_used

hec_s_count(ic)
   number of events in which channel ic had a valid signal
   if no algorithm, this is the number of events considered
```

The signal search package is controlled by the `search` datacard. The first parameter selects the search algorithm. The other 5 (real) parameters can be used by the various algorithms.

```
search i  1= search algorithms:<0= search package disabled
                               0= dummy search
                               1= standard search (D. Striegel)
                               2= (0) could be replaced by user
                                  subroutine hec_search_user
          The following values are for the standard algorithm
       r  2= timeslice to start searching
       r  3= timeslice to stop searching
       r  4= number of slices to compare (3, 5, 7)
       r  5= apply a cut on maximum
       r  6= level of the cut
```

The following algorithms can be selected:

$1 < 0$: The search package is not activated. Results are as if no valid signal was ever found.

$1 = 0$: Dummy search algorithm. Results are as if all channels always contained valid signal.

$1 = 1$: Standard algorithm (by D. Striegel). Only physics trigger events are treated by this algorithm. Let the datacard search have the following parameters:

```
2= t1  3= t2  4= ntc  5= cutflag  6= cut
```

For each channel the standard algorithm searches for the maximum adc in the timeslice range [t1, t2]

Case cutflag = 0. The time distribution is compared to a valid pulse shape. The comparison uses ntc timeslices (only the values 3, 5 and 7 are implemented).

Case cutflag = 1. If this maximum is above the value cut*rms, where rms is the channel's pedestal rms, then the time distribution is compared to a valid pulse shape. The comparison uses ntc timeslices (only the values 3, 5 and 7 are implemented).

Variables specific to this algorithm are in `hec_search_1_sys.inc`

$1 > 1$: User defined algorithm. The user can supply a routine `user_search.f` that, given an event, fills the common `hec_search_inc` following a user defined algorithm. The default user algorithm is the dummy algorithm.

Please mail any comments or questions to striegel@mppmu.mpg.de .

9.8 The System Histogram Package

System histograms can be produced by the datacard

```
mkhis  i  1= system histograms flag     (>0 = make his)
       i  2= logical unit for system histogram file writing
       i  3= do/don't (1/0) use search algorithm to determine
              which cells will fill histogram
```

For example,

```
mkhis 1=1   2=52  3=0
```

will generate the system histograms. They are listed by the hec_adc program, and they are output in the file hec_adc.hbook. Histograms produced by this package are deleted from memory after output and so do not interfere with the booking of user histograms.

The third datacard switch allows the histogram package to use the results of the search package to determine which channels will be included in each histogram on an event-by-event basis. Therefore, if this feature is activated, histograms will be filled only for the channels containing signal. For example, if 1000 events are processed and channel 72 only contains signal in 5 of these events, there will only be 5 entries in histograms referring to channel 72.

There is a second datacard (hiscon) which can be used to control the histogram package. This datacard is shared with the online package and allows the user to turn on or off the production of certain categories of histograms. This datacard is shown below:

```
hiscon  i  1= do/don't (1/0) produce pedestal and noise histograms
        i  2= do/don't (1/0) produce geometrical energy distribution histograms
        i  3= do/don't (1/0) produce uncorrected adc histograms
        i  4= do/don't (1/0) produce tdc corrected histograms
        i  5= do/don't (1/0) produce calibrated (nA) histograms
        i  6= do/don't (1/0) produce cluster histograms
```

To fill the cluster histograms, clusters must be defined. This is done with the following two datacards:

```
hisctit##
           Cluster title card. ## is the number of the cluster
           the title refers to (eg. hisctit01 is cluster 1)
        c  1= 4 character title of cluster (eg. 'pi,h')

hisclus##
           Cluster definition card. ## is the number of the cluster
           (eg. hisclus02 is cluster 2)
        i  1= number of cells in cluster (number to follow in list)
        i  2= channel number of first cell in cluster
        i  3= channel number of second cell in cluster
        i  etc. up to number of cells specified in first argument
```

For example the following datacards define one pion cluster with title "H,pi" (pion hitting impact point H in August 1998) composed of the 19 cells 3, 5, 4, 76, 1, 16, 15, 88, 14, 13, 86, 12, 11, 84, 41, 24, 23, 96 and 21:

```
hisctit01 'H,pi'
hisclus01 1=19 3 5 4 76 1 16 15 88 14 13 86 12 11 84 41 24 23 96 21
```

Similarly, the folowing datacards define one electron cluster with title "H,e-" composed of the 3 cells 3, 11 and 13:

```
hisctit02 'H,e-'
hisclus02 1=3 3 11 13
```

The histogram package now contains the same histograms available in the online display at testbeam. See section 10 for a list of the available histograms.

9.9 The Standard Ntuple Package

Communication with the ntuple package is done through the following datacard:

```
mkntp   i  1= output standard ntuple (1=yes,0=no)
        i  2= logical unit number of ntuple file
        i  3= units of energy in event ntuple
              (1=adc,2=nA,3=GeV)
        r  4= difference from expected time-zero (in ns).
              used to screen out events with bad timing
              (0. deactivates)
        i  5= turn on/off (1/0) ntuple user routines
```

The ntuple package is designed to produce three standard column-wise ntuples for each run which contain a preprocessed version of the information available in the raw data file. It is expected that an agreed-upon hec_adc datacard will be used to fill these ntuples for each run and offline analysis will be performed on the ntuples rather than on the raw datafiles. These ntuples share a common structure with the Monte Carlo production ntuples for the hadronic endcap allowing easy comparison of data and Monte Carlo. A common ntuple analysis program is forseen for data and Monte Carlo.

The three ntuples produced by the package are:

1. run header ntuple (ntuple 100)
2. event ntuple (ntuple 101)
3. slow control ntuple (ntuple 102)

the contents of which are:

```
run ntuple:
              hec_runno        i    run number
              hec_runpd        i    run period number
              hec_beame        r    beam energy
              hec_noevt        i    number of events
              hec_parttype     i    particle type
              hec_cryox        r    cryostat position in x
              hec_tabley       r    table position in y
              hec_zbeam        r    for MWPCs, z position at which x and y
                                    are calculated
              hec_peakf        i    peak finding method used
                                    1 = cubic
                                    2 = digital filtering
                                    0 = pure MC
                                   -1 = spoiled MC with ordinary noise
                                   -2 = spoiled MC with noise as after
                                        digital filtering
              hec_cal_version  i    version of calibration used
              hec_dig_version  i    version of digital filtering weights used
              hec_shower       i    shower model type
                                    0 = experimental data
                                    3 = GHEISHA
                                    4 = G-FLUKA
                                    6 = G-CALOR
              hec_eunit        i    units of energy
                                    (1=adc,2=nA,3=GeV,4=MC visible energy)

              hec_adc_max      i    maximum adc channel number (eg. 144)
              hec_index        i    index of channel with a given adc number

              hec_cells_used   i    number of cells used (eg. <= 144)
              hec_ped_rms      r    run pedestal rms for each channel
              hec_ieta         i    eta value for each channel
              hec_iphi         i    phi value for each channel
              hec_iz           i    z value for each channel
```

```
              hec_ic           i    adc channel number

 event ntuple:
        standard event block (hecevt - shared with MC)

              hec_evtno        i    event number
              hec_trig         i    trigger flag array
                                    (physics,electron,pion,muon,random)
              hec_signal       r    energy (signal) for each channel

        MWPC block (hecmwpc - if requested)
              hec_xbeam        r    beam x-coordinate at z=zbeam
              hec_ybeam        r    beam y-coordinate at z=zbeam
              hec_xslop        r    trajectory x-slop: at any z
                                    x=xbeam+xslop*(z-zbeam)
              hec_yslop        r    trajectory y-slop: at any z
                                    y=ybeam+yslop*(z-zbeam)
              hec_xbeam_ok     i    =1 if beam x-projection is successfully
                                    reconstructed
              hec_ybeam_ok     i    =1 if beam y-projection is successfully
                                    reconstructed
        user block (hecuser - if requested)
              user-defined content

 slow control ntuple: (not yet defined)
```

The content of these ntuples has been chosen assuming that ntuple analysis will be performed in dedicated analysis programs rather than in PAW. This assumption means that it is possible to tie the event ntuple to the run header ntuple in order to save space without reducing functionality. In other words, the event ntuple is not meaningful on its own, the indices and conversion arrays of the run header are necessary to give it meaning.

Since the cell numbering scheme is not consecutive (ie. some cells are dead or disconnected) it is necessary to have an index variable to allow conversion from array index to channel number, or physics system coordinate (η,ϕ,z). The run header ntuple variable `hec_index` is the index variable for this set of ntuples. This means that the value of `hec_index(2)` is the array index corresponding to adc channel number 2. Sample Fortran code is shown below which could be used in an ntuple analysis program to print the contents of a given channel.

```
c
c print signal in channel 28
c
print *,'signal in channel 28 is ',hec_signal(hec_index(28))
c
c loop over all good cells in run
c print eta of every channel, adc # of every channel and
c signal in channel 28 found by alternate method.
c
do i = 1, hec_cells_used
```

```
  print *,'ieta of channel ',hec_index(i),' is ',hec_ieta(hec_index(i))
  print *,'ic of array index ',i,' is ',hec_ic(i)
  if (hec_ic(i) .eq. 28) then
    print *,'signal in channel 28 is ',hec_signal(i)
  endif
enddo
```

This example demonstrates the use of variables from both the run header and event ntuples, making it awkward to do in PAW. However, if the user needed to do a quick check of cell 28 in PAW, the following procedure would work:

```
nt/pl 100.hec_index(28)

(now read from screen the value of the index (eg. 26) corresponding to
 channel number 28)

nt/pl 101.hec_signal(26)
```

In the future, an ntuple analysis program may be provided and knowledge of the details of this structure may not be necessary for most users.

In addition to the standard event block (hecevt) in ntuple 101 (described above) two other blocks can be added for each event at the users request. If the user activates the beam chamber reconstruction package a special MWPC block will be added for each event, the content of which is also described in the listing above. A second block may be added for each event with user-defined content. For this purpose, two user routines and a special user include file have been created. The routine `user_ntuple_block_ini.f` is called once before the ntuple event loop and can be used for booking the ntuple block. The block can be filled from the routine `user_ntuple_block_evt1.f` which is called every event. Example routines are provided in the user example directory (see Section 12) and if the user block is activated (from ntuple package datacard) a sample block will be added to the event ntuple.

10 The Online Package

In order to monitor testbeam data as it is taken, a special package has been written which allows the offline routines to be used as monitoring tools. This "online package" is activated by way of the rnmode datacard:

```
 rnmode   1= offline/online (1/2) mode (1 is default) .
```

Activating the online package overrides many of the datacards controlling the other system packages. The online package essentially replaces the main program of the offline code and calls specific routines from different system packages in order to run through the data as quickly as possible. It produces an interactive ASCII menu and a graphical display of several useful histograms.

10.1 How to Use the Online Package

- Create a link to the directory in which the data is stored. The online mode of the hec_adc package automatically looks for data in a directory called "data" in the directory from which it is called. If users wish to access data in another area (eg. on another disk) then

they must create a link to that area and name the link "data" as in the following example:
`ln -s /datadisk/datadir/ data/`
This UNIX command creates a link such that any reference to the directory "data" will be instead pointed to "/datadisk/datadir/".

- Set the maximum number of events using `nevtm` in `hec_adc.datacard`.
- Set `rnmode 1=2` in `hec_adc.datacard`.
- Type `hec_adc` on the command line
- After some initialization statements are echoed to the screen the following request will be displayed:
```
***ENTER RUN NUMBER TO MONITOR (0 to exit)
```
- Enter the number of the run to be analyzed as a 4-digit number (eg. 6733). The program will then look for a file called `./data/run_6733.dat`.
- The user should then see output similar to the following:

```
***hec_online_getfile searching for data file data/run_6733.dat
***Successfully opened data/run_6733.dat
***Opened file data/run_6733.dat  for online analysis
***Found data file data/run_6733.dat starting analysis
***Select Trigger type to analyse:
*** = 1  : all event types
*** = 2  : physics triggers
*** = 3  : electron triggers
*** = 4  : pion triggers
*** = 5  : muon triggers
```

  Now a choice must be made as to which type of events the user wishes to analyze. Only events which pass the selected trigger requirements will be included in online histograms.

- The analysis will then be performed and a summary plot will be displayed in a graphical window. In the input window a new menu will now appear, see Figure 10.1. This menu lists the available histograms. Typing the number of any of these histograms will lead to it being displayed in the graphical window. Four buttons should then appear in the upper left corner of the graphical window, one labelled "PS", one labelled "ZOOM", one labelled "GAUSS" and one labelled "ZONE".
  - Clicking on the PS button will cause the histogram to be saved in a postscript file called user_summary.ps. All histograms saved in this manner during the analysis of a single run will be stored in the same postscript file.
  - Clicking on the ZOOM button will allow the user to zoom on the displayed histogram by clicking the left mouse button on either end of the range to be zoomed on (as per the instructions in the ASCII window).
  - Clicking on the GAUSS button will fit a Gaussian curve to the currently displayed histogram, printing the parameters of the fit in the upper right corner of the display.
  - Clicking on ZONE toggles between 1 histogram/page and 4 histograms/page.

  To exit the graphical display and return control to the ASCII window click the right mouse button anywhere on the graphical window.

- Enter 0 to exit the histogram menu and the program is ready to analyze another run.
- Enter 0 to exit the program.

11 Utilities

Included with the hec_adc package is a directory containing utilities that may be of some help to the user. The directory is `hec_adc/util` and it contains both Fortran and UNIX utilities. Users are encouraged to submit their own utility routines for inclusion in this directory if they believe that the utilities would be of general use. Routines or scripts in the utility directory are not supported to the same extent as routines necessary to run the package (ie. the debugging and testing is not as thorough).

11.1 Fortran Utilities

The `util/` directory contains several Fortran routines that are compiled and included automatically in the hec_adc library and are therefore callable from the user routines. These routines are useful for printing the contents of particular data structures. For example, the directory currently contains three routines to print event information:

```
hec_dump_adc.f      for printing the contents of each adc channel
hec_dump_dwpc.f     for printing the content of the beam chamber bank
```

and one routine to print out run period information:

```
hec_dump_geo.f      for printing out the geometry correspondence tables
                    of the chosen run period
```

11.2 UNIX Utilities

Since this package is supported on a UNIX platform, many useful UNIX scripts can be written to make common tasks easier. One such tool has been written for inclusion with the package. Though these scripts attempt to provide a generally useful service, and do not require any modification in order to run, the user should feel free to modify them to serve their particular needs (particularly if the user happens to be a UNIX guru).

11.2.1 multi_run.job

This UNIX tool is meant to make it easier to run the hec_adc package on multiple runs. The name of the script is `multi_run.job` and it will link to each file in a list, run the hec_adc package, rename the log file and hbook file and move on to the next member of the list.

The script accepts three different syntaxes. The first syntax:

```
multi_run.job -firstrun -lastrun
```

will process all runs between firstrun and lastrun. For example, `multi_run.job -6100 -6103` will process runs 6100, 6101, 6102 and 6103. The second accepted syntax:

```
multi_run.job run1 run2 run3 run4 ......etc.
```

will process a run list of up to 9 runs entered on the command line. The final syntax:

```
multi_run.job 0 runlist.txt
```

```
*************************************************
* Histogram  Number  |  Content                   *
*************************************************
* 1                  |  adc pedestals             *
* 2                  |  pedestal vs. ic           *
* 3                  |  adc pedestal rms          *
* 4                  |  adc pedestal rms  vs. ic  *
* 5                  |  Max(adc) vs. ic           *
* 6,7,8              |  lego of Max in iz=1,2,3   *
* 10    + iz         |  Max-ped for depth iz      *
* 14,15              |  Depth profile: module 1,2 *
* 20                 |  TDC distribution          *
* 1000  + ic         |  Max-ped for (ic)          *
* 2000  + ic         |  Max(time) (ic)            *
* 2500  + ic         |  Max(time)+TDC in ns (ic)  *
* 5200  + chamber #  |  MWPC hit wire distribution*
* 5210  + chamber #  |  MWPC profile (cm)         *
* 10000 + ic         |  ADC vs time sample (ic)   *
* 11000 + ic         |  ADC vs time in ns (ic)    *
* 12000 + ic         |  Energy in nA (ic)         *
* 13000 + # of sigma |  Energy above # sigma cut  *
* 14000 + #          |  Energy in muon tower #    *
* 14100 + #          |  ADC - ped in muon tower # *
* 15000              |  Show clustering menu      *
* 15000 + cluster #  |  Clustered Energy (ADC/GeV)*
*************************************************
****note: ic refers to channel # *****************
*************************************************
*************************************************
* Summary Histogram  |  Content                   *
*************************************************
* 9990               |  NOISE                     *
* 9991               |  SHAPE/TIME                *
* 9992               |  ENERGY PROFILE            *
* 9993               |  HIGH ENERGY CELLS         *
* 9994               |  BEAM CHAMBERS (X)         *
* 9995               |  BEAM CHAMBERS (Y)         *
* 9996               |  ENERGY CLUSTERS           *
*************************************************
```

Figure 3: Menu listing the available histograms after online reconstruction.

will process all runs listed in a text file named in the command line. The text file should just contain a single column of run numbers as in the example:

```
6100
6104
6106
6108
```

The list may be as long as the user needs. At the top of the script the user must set the name of the directory that contains the data files and the name of the hbook file that will be retained for each run.

12 Example

There is an extra user directory called `user_example/` included with the package which contains a complete working example of a simple set of user routines. In these user routines, there are several examples of how to use variables from provided common blocks. Also, an ntuple is booked and filled with some of these variables. These routines are very well commented and should serve as a good guide to writing user routines.

13 Conclusion

The hec_adc software package allows for a robust and easy access to the Hadronic Endcap calorimeter beam test data. It has been used extensively.

The recent addition of a standard ntuple has allowed the development of high level analysis programs, more or less independently of the many hardware features that change from run period to run period.

Future development ideas include the treatment of slow control records (only event records are currently treated). Furthermore, it is hoped that the standard ntuple will ease the merging of the data in the case of a combined run with the Forward calorimeter or with the Electromagnetic Endcap calorimeter.