CLAS-NOTE-93-012

# Ec 3.1.2

FORWARD ELECTROMAGNETIC CALORIMETER
RECONSTRUCTION SOFTWARE

K.B.BEARD
HAMPTON UNIVERSITY

AUGUST 27, 1993

Ec 3.1.2 is the current release of the CLAS forward electromagnetic calorimeter (Ec) event reconstruction software. While it is definitely not the final release, it incorporates all essential functions and required data structures.

# 1 Introduction

The CEBAF Large Angle Spectrometer (CLAS) consists of multiple detector packages; each package requires unique software for event reconstruction. To successfully produce an integrated analysis environment, these packages must cooperate with one another and conform to a common set of standards.

The Ec package was conceived to link into some sort of yet unspecified host program; the **Ec.exe** program allows nearly all functions to be exercised one event at a time, while **Ec_engineC.exe** program calls those routines from a simple shell. The latter's virtue is to test the Ec package on a large number of GEANT created events. The only data input to Ec is via CODA-CLAS 2.0 event buffers [1] , the only output is via the *EcAcc* (EC ACCess) routines or the *EcRsl* (EC ReSuLts) common blocks.

# 2 Important Concepts

## 2.1 Philosophy

The goal of the Ec package is to accomplish correct reconstruction of CLAS forward electromagnetic calorimeter events using clean, structured, maintainable, and documented code. Conceptually, information at each stage of analysis is stored in common blocks; the subroutines are operators used on one common to fill another; and only instructions and status information are passed as arguments. The parameters are carefully chosen and ordered to characterize all information about objects within commons, and the commons constructed to organize the analysis. This structure may be easily modified by adding or deleting parameters.

Because Ec must work in concert with other packages being developed independently, it is modularly structured: various parts (algorithms, graphics, interfaces, simulations) may be substituted for by dummy subroutines without recoding.

[1] CLAS Event Format, Version 2.00, CLAS-NOTE 93-002, March 24, 1993

## 2.2 Coding Comments

Ec has been written to conform to the CLAS software standards [2] with very few exceptions. Standardized nomenclature has been used throughout Ec; the first two letters identify the package, the next three the functionality. In the absence of a SwGen package of general purpose routines, the personal *KB_library* of routines were used; these are supported under DEC-Ultrix and DEC-VMS, but not yet under HP-UX and IBM-AIX. Future releases will use the SwGen routines.

The Ec package compiles and runs correctly under DEC-VMS and DEC-Ultrix; all details specific to the platform are hidden in the *KB_library* libraries. They consists of *KBB_tricks* and *ULTRIX_specific* (Ultrix) or *VMS_specific* (VMS).

## 2.3 Terminology and Units

For the purpose of Ec, many details of the detectors construction are ignored. Only the active region is considered, and the detector is considered homogenous. Each calorimeter is divided into an *inner* and an *outer layer*; the volume of space viewed by a single phototube is called a *strip*. The sides of the calorimeter are denoted *U, V,* and *W*. The shortest strip is numbered 1. The Ec package uses the following name conventions; a *pixel* is the volume formed by the intersection of a U, V, and W strip. A *hit* is a reconstruction of energy deposited which was read by intersecting U, V, and W strips within a layer. A *shower* is formed by the overlap (in local IJ space) of an inner and outer hit. Note that such a shower may represent an electromagnetic shower, a minimum ionizing path, or any event depositing energy in both layers. Events corresponding to energy which is measured in a strip, but cannot be represented by intersecting U, V, and W strips, are called *unrecon*, short for unreconstructable.

The CLAS XYZ coordinate system is right handed with Z parallel to the beam axis, X horizontal through the center of sector 1, and the origin at the nominal target position. The CLAS spherical system is the normal complement to the XYZ system.

[2] The Hitchhiker's Guide to the Galaxy: CLAS Software Manual (Rev 1.0), CLAS-NOTE-90-008, July 16, 1990

The sector coordinate system S123 is right handed and sector dependent with S1 parallel to the beam and S2 through the center of the sector.

The local right handed coordinate system is IJK and has its origin such that K is normal into the inner face of the calorimeter and parallel to the radial vector from the target. I is away from the beam in the plane of the inner face of the calorimeter. The local UVW system is the nonorthogonal system measured along the edge of the calorimeter.

All units are the CLAS standard; angles in radians, energy in GeV, distances in centimeters, and time in nanoseconds.

## 2.4 Error Messages

The Ec package is still being developed; in particular the reconstruction algorithms are being improved and made more robust with respect to missing information (dead photomultiplier channels) and much faster. This release carries a considerable burden of error checking; as the correctness of the code is verified these routines will gradually be removed in newer releases. The *OK* status returned by most subroutines and the *err* messages are primarily to detect flawed code; the path to the error as well as the error is returned. In general, correct code should never produce an *OK=.FALSE.* status.

# 3 Input

The only supported data inputs are CODA-CLAS 2.0 event buffers. Commands may be issued directly by calling subroutines or by using the *pseudo_Ec* interface. **Ec.exe** and **Ec_engineC.exe** are examples of both.

The interactive interface is one taken from the *pseudoQ* package [3]; it is allows nested command files and is generally very similar to the LAMPF Q system interface [4].

It is not necessary to use the pseudoQ interfaces at all; they are provided merely as a convenient way to communicate with the package.

[3] CAMAC_chat, A CAMAC Communication and Software Development Tool, CLAS-NOTE-91-026, January 14, 1992

[4] Q Release Notes and Distribution Information, Release: March 17, 1990, Document MP-1-3413-6, Los Alamos National Laboratory.

## 3.1 pseudoQ Syntax

All the interactive interfaces are called in the form:

```
pseudo_EcXxx(input_line,done)
```

where **Xxx** is the subpackage interface name. The subroutine *pseudo_Ec* can call all other subpackage interfaces.

All pseudoQ commands of the form:

```
FIELD1/op1 /op2/op3:n FIELD2/op4:string
```

Options are always preceeded by a "/" and may be followed by numbers or a string (separated by ":"). The relative order of the options and fields is ignored. The "::" denotes a sequence (1::5 is the same as 1:2:3:4:5). The "!" or ";" begins a comment field; anything beyond that point is ignored. A line is continued on the next line if it terminates in a "&". The fields usually provide information for the options. Commands are not case sensitive but the case of file names is preserved for UNIX compatability.

In the main shell (EC) the desired routine may be specified by the first field. If only the field is specified, the corresponding subroutine (pseudo_EcXxx) is entered and its prompt given; otherwise the single line is processed and the routine exited. The routine is exited by an End-of-File character or a "%".

PseudoQ command files are specified as "@filename".

# 4 Output

The **Ec_engineC.exe** is an example of using the (EC ACCess) routines and putting the results into an CERNLIB Ntuple [5]. The interactive routine *pseudo_EcAcc* will report type and quantity of information available on specific objects within a common. It can be used to build a *vector*; a list of

[5] HBOOK User Guide, Version 4, Y250, October 28, 1987, CERN Computer Centre Programming Library

characteristics of a specified object within a common that can subsequently be packed into any user array. This technique decouples details of the Ec package (the ordering of characteristics, for example) from the host code. **Ec_engineC.exe** extracts information from the simulated event descriptor block, calculates a few quantities based on that, and packs the resulting data and concatenates all vectors to form a single CERNLIB Ntuple. The resultant Ntuple can be used by PAW [6]. The interactive routine to project histograms and make cuts.

The **only** direct access to any Ec common should be the *EcRsl* (EC ReSuLts) commons. The include files are "EcRsl_export.PAR", "EcRsl_status.CMN", "EcRsl_export.CMN", and "EcRsl_par_desc.CMN", they define the information content, structure, meaning, and status of the output results. The status should always be checked before the results are used.

## 4.1 Graphics

The *EcGra* (EC GRAphics) subpackage was developed to support software development, and be compatible with a general purpose host. It is based on the CERNLIB HIGZ package [7] and assumes the set window scale is the physical scale (all scaling, offsets, etc. are done by HIGZ behind the scenes). In this way various packages' graphics can be easily superimposed without communication between the packages. The display options are preselected by a call to *pseudo_EcGra* or *EcGra_option.*

The graphics are fully three dimensional; the CLAS XYZ, sector S123, and Ec IJK coordinate systems are supported as are rotations about the vertical and horizontal axes; hidden lines and color are not yet supported. Sectors, layers, and objects may be plotted independently. A common energy scale is used throughout but may be changed at any time. The call is of the form:

```
        call EcGra_plot(layer,sector)
```

For both layers and all sectors, specify *layer=0* and *sector=0.*

[6] PAW - Physics Analysis Workstation, The Complete Reference, Version 1.07, Q121, October 1989, CERN Programming Library

[7] HIGZ - High level Interface to Graphics and Zebra, Q120, March 10, 1988, CERN Computer Centre Programming Library

# 5 Simulations

This release reads and writes CODA-CLAS 2.0 data files using the EvGen 2.0 package routines. The internal simulations are crude parameterizations and are for internal testing **only**; detailed GEANT simulations [8] should be used to produce realistic events. The current GEANT release for CLAS, *CLAS315* [9], writes events in CODA-CLAS 2.0 format and counts photomultipliers the same way as Ec and the software standard, but disagrees slightly with Ec on the location of the calorimeters.

# 6 Reconstruction

There are three reconstruction algorithms supported in this version; the first is **Quick** and takes $\sim 0.5$ MI but has no attenuation correction. Next is **Fast** and requires $\sim 2.5$ MI, last is the **Edge** and requires $\sim 5.0$ MI.

The **Quick** algorithm forms clusters of strips on an edge within a layer, then determines possible hits from the spatial overlap. It will report disconnected clusters as overlapping hits. It calculates neither peaks nor pixels.

The **Fast** algorithm assigns pixels energies based on the strips; it then forms clusters of pixels and thence hits. It is the initial step of an unreleased Metropolis algorithm.

The **Edge** algorithm takes the signals from the strips on a single edge (U, V, or W) and groups them into *peaks*. Peaks from each edge are combined such that the geometric constraints are met to form hits. Hits from the inner and outer layer are combined to form showers. Pixels are not used in this algorithm, but are calculated subsequently for comparison and display.

## 6.1 Linking

The Ec package can be linked as a stand alone program (**Ec.exe**) or as part of a larger package; with or without graphics; and with or without pixels. Dummy routines are provided so Ec need not be modified if a host contains a CERNLIB PAWC common and performs the necessary initializations of the

[8] GEANT Simulation of CLAS Forward Calorimeter Performance, CLAS-NOTE 93-009, August 17, 1993

[9] GEANT Simulation Program for CLAS,CLAS-NOTE 93-013, August 23, 1993

memory, workstation, and graphics. The general Ec library includes both pixels and graphics; the PAWC common may be initialized by calling the appropriate routines.

# A General Ec Routines

There are only a few routines the average Ec user need call; they are described briefly here. Results should be accessed only as described in Appendix B. Success or failure is returned in *OK* and a reason for failure in *err*.

*Subroutine* Ec_initialize_all ( OK, err)
*Character*(*)* err
*Logical* OK

This subroutine initializes all the geometric and calibration information except graphics used by Ec.

*Subroutine* EcGus_version_report ( description,length)
*Character*(*)* description
*Integer* length

A brief *description* of the current version of Ec is returned as well as its nonblank *length*.

*Subroutine* Ec_reset_all ( OK, err)
*Character*(*)* err
*Logical* OK

This subroutine resets status flags and counts for data from a previous event but saves time by not clearing all arrays.

*Subroutine* Ec_store_all ( index, buffer, pointers, OK, err)
*Integer* index, buffer(*), pointers(*)
*Character*(*)* err
*Logical* OK

Given the *index* (typically 1), *buffer*, and *pointers* from *EvGen_get_event*, this routine copies the TDC and ADC values into the EcEvu common in preparation for analysis.

*Subroutine* EcGus_check_status ( common, ready, sector, OK, err)
*Character*(*)* common, err
*Logical* ready, OK
*Integer* sector

For a specific *sector* and a particular *common*, the status *ready* is returned. For example, if *common*="EcEvu" and no valid **Ec E**vent **U**npacked data exists in *sector*, but *sector* exists and has been initialized, then *ready*=.FALSE., *OK*=.TRUE., and *err*=" ".

*Subroutine* EcFit_analyze ( method,sector,OK,err)
*Character*(*)* method, err
*Integer* sector
*Logical* OK

Given a particular *sector*, this routine attempts to reconstruct the (previously stored) event using technique *method*. If *method* is blank, the last value of *method* is used; currently supported values are "Quick", "Fast", and "Edge".

# B EcAcc Access Routines

All access to objects within Ec common blocks should be made through the use of these routines; in this way can the user be sure all error checking is properly performed and that all returned information is valid. Other packages should **never** access Ec commons directly; any attempt to do so will hobble both host and Ec code development. Success or failure is put into *OK*; an explanation for any failure is put into *err*.

## B.1 pseudo_EcAcc interface

The *pseudo_EcAcc* routine was designed as an aid to code development and to provide a simple way of creating vectors and extracting data. All the *pseudo_EcXxx* routines support the "/HELP" option:

```
ECACC> /help
  EcAcc>           reconstructed information
       /HElp                list options
       /SECtor:s            CLAS sector
       /INFOrmation         list available info on object
       /AVailable           get number of objects
       /Nth:n               on Nth object in class
       /ID:id               on object id in class
       /CHaracteristics     list reqd characteristics
         :desc                  add "desc" to list
       /-CHaracteristics    cancel list
       /SPECification           show current specification
         :spec              set new "spec"
       /-SPECification          cancel specification
       /OBJect              show current object
         :obj                   set new "obj"
       /-OBJect             cancel object
       /COMmon              show current common
         :cmn                   set new "common"
       /-COMmon             cancel common
    name/BUIld              build an access vector
    name/FILL_Nth:nth       fill nth access vector
    name/FILL_Id:ID          fill id access vector
        /LIst                list all access vectors
```

The following is an example of how to examine available information without resorting to external documentation. This could be done automatically by the host code, allowing the host program to learn of changes to the Ec package.

```
EC> access                                  !call pseudo_EcAcc
ECACC>  /-char /-spec /-obj /-common !cancel any&all previous settings
ECACC> /common:? /info                      !commons currently available?
   common: ?

ERROR: EcAcc_information:HELP requested
   commons: EcFit-EcDrv
   object:
   specifications:

ECACC> /common:ECDRV /obj:? /info   !objects in common "EcDrv"?
   common: EcDrv
   object: ?

ERROR: EcAcc_information>EcDrv_information:HELP requested
 common: EcDrv
 object: SHOWERs-HITs-UNREConstructed
 specifications:

ECACC> /object:HIT /info  !spec's and char.'s allowed on object "HIT"?
 object: HIT

 common: EcDrv
 object: HIT
 specifications: INNER-OUTER
 characteristics:
           R
           THETA
           PHI
           X
```

```
            Y
            Z
            S1
            S2
            S3
            I
            J
            K
            U
            V
            W
            ENERGY
            TIME
            WIDTH
            QUALITY
            UNRECONSTRUCTED
            DARK
            DISTANCE
            ATTENUATION
            FEYNESS
            DIMNESS
            LAST_PARM

ECACC>  %<return>           !exit pseudo_EcAcc
EC>
```

An *EcAcc* command file containing the lines:

```
      /-char/-spec/-obj/-com       !cancel any previous settings
      /common:ECDRV                !set which "common"

!-select characteristics of interest
      /char:I/char:J/char:ENERGY/char:TIME   !add I,J,ENERGY,TIME to list
      /char:WIDTH/char:THETA/char:PHI        !add WIDTH,THETA, and PHI

      /object:HIT                   !set which "object"
      /spec:INNER INHIT/build       !set "specification", create vector "INHIT"
```

```
    /specif:OUTER                !set "specification"
       OUTHIT/build              !create vector "OUTHIT"

/-spec /obj:SHOWER               !unset "specification", select object
/-char /char:ENERGY              !reset "characteristics" list
 SHWR/build                      !create vector "SHWR"
```

would produce three vectors named “INHIT”, “OUTHIT”, and “SHWR”.
All defined vectors may be listed:

```
ECACC> /list
  INHIT
            INNER HIT ECDRV
         21  I
         22  J
         27  ENERGY
         28  TIME
         29  WIDTH
         13  THETA
         14  PHI
  OUTHIT
            OUTER HIT ECDRV
         21  I
         22  J
         27  ENERGY
         28  TIME
         29  WIDTH
         13  THETA
         14  PHI
  SHWR
              SHOWER ECDRV
         27  ENERGY
```

## B.2 EcAcc Routines

The complete set of *EcAcc* routines are described, but a typical user would probably only call *pseudo_EcAcc*, *EcAcc_find_Nvectors*, and *EcAcc_fill_Nth.*

*Subroutine* pseudo_EcAcc ( command,OK)
*Character*(*)* command
*Logical* OK

This pseudoQ interface allows most *EcAcc* routines to be called. If *command* is blank, the pseudoQ interface goes into interactive mode, otherwise the command or command file is executed and the routine returns. Interactive mode is terminated by an End-of-File character or a leading “%”. Command files are immediatly preceeded by a “@”, and may be nested.

*Subroutine* EcAcc_information ( specification, object, common, Nchr, chr, OK, err)
*Character*(*)* specification, object, common, chr(*), err
*Integer* Nchr
*Logical* OK

This subroutine returns information about an *object* in the Ec *common.* If the class of *object* exists, the *specification* returned contains all allowed values for selecting a particular set objects within a sector, and contains the range of each dimension allowed for that object. A “.” separates required fields; “-” the possible values of each field. For example, for *common*=“EcDrv_general”, *object*=“HIT”, the returned *specification* is “INNER-OUTER”, for *object*=“UNREC” the returned *specification* is “LOW-MEAN-HIGH.INNER-OUTER”. Hence, “OUTER” “HIT” is a hit in the outer layer, and “HIGH INNER” “UNREC” gives an upper limit on characteristics of an unreconstructed object in the inner layer. All information about an object consists of characteristics; the number of supported characteristics (*Nchr*) is returned and a character array (*chr(*)*) filled with a brief ASCII label for each characteristic. For *common*=“EcDrv_general”, *object*=“SHOWER”, there are 23 characteristics, beginning with “R”,“THETA”,“PHI”,... and running to “END of parameters”. Setting *common*=“?” will produce a list of valid commons; *object*=“?” will produce a list of valid objects within a common.

*Subroutine* EcAcc_Nfound ( specification, object, common, Nfnd, sector, OK, err)

Character*(*) specification, object, common, err
*Integer* Nfnd, sector
*Logical* OK

This subroutine returns the number of specified objects in the requested *common* for a particular *sector.* If the *specification*, *object*, *common*, and *sector* are valid (ex: "INNER","HIT","EcDrv_general",3), the status is checked and the number of objects present is put into *Nfnd.*

*Subroutine* EcAcc_request_Nth ( Nth, specification, object, common, ID, Nchr, chr, info, sector, OK, err)
*Character*(*)* specification, object, common, chr(*), err
*Integer* Nth, ID, Nchr, sector
*Real* info(*)
*Logical* OK

This subroutine fills the array *info* with the values of requested characteristics (*chr(*)*) of the *Nth* specified object of the selected *common*. Checks are made that the *Nth* specified object within the common and the characteristics thereon exist. For example, for the 1st (*Nth*=1) specified object (*specification*="INNER",*object*="HIT") in the sixth sector (*sector*=6), a request for only the energy (*Nchr*=1,*chr(1)*="ENERGY") would fill *info(1)* with the energy of that particular object. The *ID* is the returned internal Ec ID number for the object.

*Subroutine* EcAcc_request_ID ( ID, specification, object, common, Nth, Nchr, chr, info, sector, OK, err)
*Character*(*)* specification, object, common, chr(*), err
*Integer* Nth, ID, Nchr, sector
*Real* info(*)
*Logical* OK

Similar to *EcAcc_request_Nth*, this subroutine uses the *ID* number rather than the index *Nth* number.

*Subroutine* EcAcc_build_vector ( name, specific, object, common, Nchar, char, OK,err)
*Character*(*)* name, specific, object, common, char(*), err
*Integer* Nchar
*Logical* OK

This subroutine defines a vector *name* to be a list of *Nchar* characteristics *char* associated with a *specific object* within a given *common*.

*Subroutine* EcAcc_find_Nvectors ( name, Nfnd, sector, OK, err)
*Character*(*)* name, err
*Integer* Nfnd, sector
*Logical* OK

This subroutine returns the number of objects *Nfnd* described by the previously defined (by *EcAcc_build_vector*) vector *name* within a *sector*.

*Subroutine* EcAcc_fill_Nth ( Nth, name, ID, local, sector, OK, err)
*Integer* Nth, ID, sector
*Character**(*)* name, err
*Real* local(*)
*Logical* OK

This subroutine fills array *local* with the using the previously defined (by *EcAcc_build_vector*) vector *name* associated with the *Nth* object. The *ID* of the object is also returned.

*Subroutine* EcAcc_fill_ID ( ID, name, Nth, local, sector, OK, err)
*Integer* ID, Nth, sector
*Character**(*)* name, err
*Real* local(*)
*Logical* OK

This subroutine fills array *local* with the using the previously defined (by *EcAcc_build_vector*) vector *name* associated with object number *ID*. The sequential ranking *Nth* of the object is also returned.

# C Ec common blocks

The electron calorimeter (Ec) reconstruction software encompasses several stages of analysis and processes each sector separately. All information other than controls and status are passes via common blocks. In turn these are defined using carefully selected parameters; hence these parameters define the information content of the Ec package. Each step of analysis is characterized by separate common blocks:

| DESCRIPTION | COMMON name |
|---|---|
| raw event block: | EcEvb |
| unpacked raw event: | EcEvu |
| calibrated strips: | EcCal |
| edge fitting algorithm: | EcFit_edge |
| metropolis fitting algorithm: | EcFit_pixels |
| general fitted results: | EcFit_general |
| best reconstruction results: | EcDrv_general |
| general status information: | EcGus_status |

Only *EcRsl* files should ever be included into routines outside the Ec package. Note that the *EcRsl* parameters are different from those used by the other Ec commons.

| | |
|---|---|
| exported results: | EcRsl_export |

# D Ec parameters

The Ec package uses parameters to define the relevant characteristics of objects; not all objects use all parameters. While these parameters are only for internal Ec use, their descriptions are included here and in *Ec_par_desc.CMN*.

| parameter | label | description |
|---|---|---|
| Ec_undefined | UNDEFINED | reserved undefined Ec parameter |
| Ec_ADC | ADC | raw ADC value |
| Ec_TDC | TDC | raw TDC value |
| Ec_nearest | NEAREST | nearest geometrically to PMT |
| Ec_midpoint | MIDPOINT | geometric midpoint from PMT |
| Ec_furthest | FURTHEST | furthest geometrically from PMT |
| Ec_lowest | LOWEST | lowest numeric value |
| Ec_mean | MEAN | numeric mean |
| Ec_highest | HIGHEST | highest numeric value |
| Ec_inner | INNER | Ec inner layer |
| Ec_outer | OUTER | Ec outer layer |
| Ec_total | TOTAL | Ec cover/total of inner and outer layer |
| Ec_r | R | CLAS r spherical coord. (cm) |
| Ec_theta | THETA | theta (radians) |
| Ec_phi | PHI | phi (radians) |
| Ec_x | X | CLAS X rectangular coord. (cm) |
| Ec_y | Y | Y |
| Ec_z | Z | Z |
| Ec_s1 | S1 | local sector S1 rectangular coord. (cm) |
| Ec_s2 | S2 | S2 |
| Ec_s3 | S3 | S3 |

| | | |
|---|---|---|
| Ec_i | I | local Ec I rectangular coord. (cm) |
| Ec_j | J | J |
| Ec_k | K | K |
| Ec_u | U | local Ec U edge coord. (cm) |
| Ec_v | V | V |
| Ec_w | W | W |
| Ec_energy | ENERGY | energy (GeV) |
| Ec_time | TIME | time (nS) |
| Ec_width | WIDTH | width (cm) |
| Ec_quality | QUALITY | quality |
| Ec_unrecon | UNRECONSTRUCTED | unreconstructed energy (GeV) |
| Ec_dark | DARK | unseen energy (GeV) |
| Ec_distance | DISTANCE | distance (cm) |
| Ec_attenuation | ATTENUATION | fractional attenuation |
| Ec_feyness | FEYNESS | fractional number of dark channels |
| Ec_dimness | DIMNESS | fractional efficiency of channels |
| Ec_last_par | LAST_PARM | end of Ec parameter list marker |

# E EcRsl parameters

The *EcRsl* commons use these parameters to identify the relevant characteristics of objects; note that they are **not** the same as the Ec parameters. These parameters are required when using the *EcRsl* commons directly. Only the include files *EcRsl_export.PAR*, *EcRsl_export.CMN*, *EcRsl_status.CMN*, and *EcRsl_par_desc.CMN* should be used when accessing the *EcRsl* commons directly.

| parameter | description |
|---|---|
| EcRsl_undefined | reserved undefined EcRsl parameter |
| EcRsl_nearest | nearest geometrically to PMT |
| EcRsl_middle | geometric middle |
| EcRsl_furthest | furthest geometrically from PMT |
| EcRsl_lowest | lowest numeric value |
| EcRsl_mean | numeric mean |
| EcRsl_highest | highest numeric value |
| EcRsl_inner | Ec inner layer |
| EcRsl_outer | Ec outer layer |
| EcRsl_cover | Ec cover/total of inner and outer layer |
| EcRsl_r | CLAS r spherical coord. (cm) |
| EcRsl_theta | theta (radians) |
| EcRsl_phi | phi (radians) |
| EcRsl_x | CLAS X rectangular coord. (cm) |
| EcRsl_y | Y |
| EcRsl_z | Z |
| EcRsl_energy | energy (GeV) |
| EcRsl_time | time (nS) |
| EcRsl_width | width (cm) |
| EcRsl_quality | quality |

| | |
|---|---|
| EcRsl_dark | unseen energy (GeV) |
| EcRsl_last_par | unused last EcRsl parameter |

# F EcCal Interface

All calibration constants may be accessed or altered using the *pseudo_EcCal* subroutine. It accepts command files or interactive instructions exactly like all the other *pseudoQ* interfaces.

```
ECCAL> /help
  EcCal>                 calibration constants
        /HElp                   list options
        /SECtor:n               set shower number
        /LAYer:m                Ec layer (INNER or OUTER)
        /AXIS:desc              Ec axis (U,V, or W)
        /UNITs:scale            energy units (eV,KeV,MeV,GeV)
        /E0:value               energy pedestal
        /Ech:value              energy/channel slope
        /T0:value               time(nS) pedestal
        /Tch:value              time(nS)/channel slope
        /ATTenuation:length     attenuation_length(cm)
        /IDs[:id]               show strip info.
        /SET[:id]               set strip info.

ECCAL>
```

The following sets a universal calibrations of 3.448 MeV/channel and a 400 cm attenuation length, then examines the settings for a specific strip.

```
ECCAL> /-layer /-axis /-sector         !deselect all
 layer:INNER - OUTER
 axis: U                          - W
 sector:           1 -              6

ECCAL> /Tch:0.1 /atten:400. /unit:MeV /Ech:3.448 !set values
 Tch:    0.1000000    nS/channel
 attenuation length:    400.0000     cm
 Ech:    3.4480002E-03 GeV/channel
```

```
ECCAL> /SET:1::36                           !set all PMT channels
ECCAL> /lay:INNER /axis:W /sec:3 /ID:13     !inquire on one PMT
 layer:INNER - INNER
 axis: W                         - W
 sector:            3 -            3
       id axis layer sector     Eo     Ech     To   Tch    atten. length
       13     W INNER  3 0.000000 0.003448  0.000  0.100      400.000

ECCAL>
```

# G EcGra Graphics

The *EcGra* graphics are not an essential feature of Ec; it was created to aid code develoment and debugging. All features may be exercised using the *pseudo_EcGra* interface.

## G.1 EcGra Interface

The interface supports the "/HELP" option:

```
ECGRA> /help
   EcGra> instruction/option
         /HElp                     list options
         /DEMOnstration            example of vertical rotation
              :n                       steps
         /PLot                     generate picture
  [file] /MEtafile:n               open type n metafile
         /SECtor:n                 sector number (0=all)
         /LAYer:n                  layer number (0=all)
         /LIMit:x1:x2:y1:y2        picture limits*
         /Number:n                 number of arguements*
         /IDs:m                    id(s) list
         /VALue:r                  real value(s)
         /DEGrees:r                degrees for value(s)
         /RESet                    reset window
         /CLear                    clear
         /WOrkstation              specify workstation type
         /THReshold:v              min. threshold in MeV*
         /PIXel                    pixel above threshold*
         /HITs                     hits above threshold*
         /SHOWers                  showers above threshold*
         /PEAKs                    peaks above threshold*
         /STRIPs                   strips above threshold*

 ECGRA>
```

In general, the fields are treated as inputs to the *EcGra_option* subroutine, while options are interpreted by *pseudo_EcGra*. Fields are sent to *EcGra_option* sequentially. A "?" or "HELP" field causes a list of options to be output.

```
ECGRA> ?
 EcGra_options: (use UPPER CASE)
       "XHORZ"
       "XVERT"
       "YHORZ"
       "YVERT"
       "ZHORZ"
       "ZVERT"
       "1HORZ"
       "1VERT"
       "2HORZ"
       "2VERT"
       "3HORZ"
       "3VERT"
       "IHORZ"
       "IVERT"
       "JHORZ"
       "JVERT"
       "KHORZ"
       "KVERT"
       "3D"
       "HIST"
       "STRIPU"
       "STRIPV"
       "STRIPW"
       "STRIP"
       "HIT"
       "THRESH"
       "SCALE"
       "PEAK"
       "ROTVERT"
       "ROTHORZ"
       "ROT"
       "PIXEL"
       "SHOWER"

       "Default" - set default values
       "ECHOing" - echos requests to screen

ECGRA>
```

The CLAS XYZ, sector S123, and Ec local IJK coordinate systems are supported. Objects may be displayed using their physical extent ("3D") or proportional to their energy content ("HIST"). All selections may be set sector by sector, layer by layer. A common energy scale is used throughout. Types of objects currently supported are "SHOWER", "HIT", "PEAK", "PIXEL", and "STRIPU", "STRIPV", and "STRIPW". "STRIP" is shorthand for "STRIPU STRIPV STRIPW". To display the pixels above 1 MeV in the inner layer of sector 2 for example:

```
ECGRA>
ECGRA> -XHORZ YVERT              !<set option> set CLAS X to left, Y up
ECGRA> THRESH/val:0.001          !<set option> threshold for display to 0.001 Ge
ECGRA> /sec:2/layer:1            !consider only inner layer of sector 2
ECGRA> -STRIP -PEAK              !<set option> show no strips, no peaks
ECGRA> PIXEL/N:0                 !<set option> only show pixels above threshold
ECGRA> /sec:0 /lay:0 /plot       !all sectors, all layers - plot now
ECGRA>
```

## G.2 EcGra Routines

Only these two routines are generally needed by the user.

*Subroutine* EcGra_option ( request,N,ids,vals,layer,sector)
*Character*(*)* request
*Integer* N, ids(*), layer, sector
*Real* vals(*)

This subroutine sets graphics options used by *EcGra_plot*. The *request* is case insensitive; the list *ids* and *vals* of length *N*, *layer*, and *sector* provide extra information which may be required by the option. The option "DEFAULT" is the sets CLAS X to the left ("-XHORZ"), Y up ("YVERT"), 3d mode ("3D"), no rotation ("-ROT"), no peaks ("-PEAK"), no strips ("-STRIP"), only pixels ("PIXEL"), hits ("HIT"), and showers ("SHOWER") above threshold ("/N:0") with an energy threshold of 10 MeV ("THRESH/VAL:0.010") and an energy scale of 10. cm/GeV ("SCALE/VAL:0.010"). A request="?" or "HELP" will send a list of options to the default output device. Option "ECHO" will echo requests to the output device. No error messages are returned unless echoing is enabled. Color selection is not yet supported.

*Subroutine* EcGra_plot ( layer,sector)
*Integer* layer, sector

This subroutine sets uses HIGZ calls to display a given *layer* (INNER=1, OUTER=2, both=0) and *sector* (sector=1-6, all=0) using graphics options previously set by *EcGra_option*. It assumes the HIGZ window exists and uses the physical dimensions of the CLAS detector.

# H Ec_engineC Analysis Engine

A simple analysis engine, *Ec_engineC.exe*, demonstrates the use of the Ec package. A single call

```
call Ec_initialize_all(OK,err)
```

initializes all of the Ec package geometry and calibrations. Before each event,

```
call Ec_reset_all(OK,err)
```

prepares the package for the next event. The standard *EvGen_get_event* subroutine provides a CODA-CLAS 2.0 format buffer and associated pointers, which are passed to:

```
call EcEvu_store_all(index,buffer,pointers,OK,err)
```

which stores the TDC and ADC values, then uses

```
call EcGus_check_status('Evu',done,sector,OK,err)
```

to see if there is any useful (EVent Unpacked) data stored in the sector, and if so then uses

```
call EcFit_analyze(method,sector,OK,err)
```

on that sector. The number of showers reconstructed is accessible by using a previously defined vector:

```
call EcAcc_find_Nvectors('SHWR',N,sector,OK,err)
```

and previously requested characteristics of the 1st and largest shower by:

```
call EcAcc_fill_Nth(1,'SHWR',ID,local,sector,OK,err)
```

Then *Ec_engineC.exe* stores the returned characteristics as part of a CERNLIB Ntuple for subsequent analysis by PAW.

## H.1 Ec_engineC example

If the command file from Appendix B is used, **Ec_engineC.exe** will then produce the following Ntuple, ID=1001 and title "INHIT+OUTHIT+SHWR". Quantities 2-6 are taken from the simulated event descriptor block, and 7-9 calculated from it.

```
********************************************************
* NTUPLE ID= 1001  ENTRIES=   6000   INHIT+OUTHIT+SHWR
********************************************************
*  Var numb  *   Name    *    Lower     *    Upper     *
********************************************************
*      1     * event ID  * -.999990E+05 * 0.100000E+04 *
*      2     * Ld id     * -.999990E+05 * 0.110000E+02 *
*      3     * Ld mass   * -.999990E+05 * 0.511000E-03 *
*      4     * Ld Q      * -.999990E+05 * -.100000E+01 *
*      5     * Ld theta  * -.999990E+05 * 0.436344E+00 *
*      6     * Ld phi    * -.999990E+05 * 0.000000E+00 *
*      7     * Ld Etot   * -.999990E+05 * 0.100000E+01 *
*      8     * Ld I      * -.999990E+05 * 0.587106E-02 *
*      9     * Ld J      * -.999990E+05 * -.114207E-04 *
*     10     * sector    * -.999990E+05 * 0.100000E+01 *
*     11     * I INHIT   * -.999990E+05 * 0.999990E+05 *
*     12     * J INHIT   * -.999990E+05 * 0.999990E+05 *
*     13     * ENERGY I  * -.999990E+05 * 0.999990E+05 *
*     14     * TIME INH  * -.999990E+05 * 0.999990E+05 *
*     15     * WIDTH IN  * -.999990E+05 * 0.999990E+05 *
*     16     * THETA IN  * -.999990E+05 * 0.999990E+05 *
*     17     * PHI INHI  * -.999990E+05 * 0.999990E+05 *
*     18     * I OUTHIT  * -.999990E+05 * 0.999990E+05 *
*     19     * J OUTHIT  * -.999990E+05 * 0.999990E+05 *
*     20     * ENERGY O  * -.999990E+05 * 0.999990E+05 *
*     21     * TIME OUT  * -.999990E+05 * 0.999990E+05 *
*     22     * WIDTH OU  * -.999990E+05 * 0.999990E+05 *
*     23     * THETA OU  * -.999990E+05 * 0.999990E+05 *
*     24     * PHI OUTH  * -.999990E+05 * 0.999990E+05 *
*     25     * ENERGY S  * 0.575816E+00 * 0.999990E+05 *
********************************************************
```

A typical *Ec_engineC.exe* session that reads the GEANT generated CODA-CLAS 2.0 file "/clas01/usr/users/beard/GEANT/e.evt", use **Edge** reconstruction, and has vectors and calibrations specified in "all.COM" follows. The options unique to *Ec_engineC.exe* are SHOW, whether to display each event, PLOTERROR, whether to display those events which return errors, HALT, whether to stop and call *pseudo_Ec* when an error is reported, and EXIT, the immediate termination of the program.

```
        Ec_engine..............K.B.Beard 6/93

  Crude analysis engine for testing the Ec package
  using CLAS 2.0 format


  >>>for testing of Ec package-  type /HELP for help

 List of valid workstation types:
       0:  Alphanumeric terminal
    1-10:  Describe in file higz_windows.dat
  n.host:  Open the display on host (1 < n < 10)
       m:  PAW_MOTIF on local host
  m.host:  PAW_MOTIF on specified host
    7878:  FALCO terminal
    7879:  xterm

 Metafile workstation types:
    -111:  HIGZ/PostScript (Portrait)
    -112:  HIGZ/PostScript (Landscape)
    -113:  HIGZ/Encapsulated PostScript
  -777/8:  HIGZ/LaTex

  workstation type?
0
```

```
  CLAS 2.0 format event file?:
/clas01/usr/users/beard/GEANT/e.evt
  event file opened OK

  Log output file?:
e_edge
  log file opened OK

  Options:[SHOWall,PLOTerrors,HALT,EXIT]
 Options:> -show
 Options:> -ploterr, -halt
 Options:> method= EDGE
  reconstruction method: EDGE
 Options:>

 EC> @all.COM                !create vectors, set calibrations

  mtup=           1
  title:INHIT
           1  INHIT                               11
           1        11  I INHIT
           1        12  J INHIT
           1        13  ENERGY INHIT
           1        14  TIME INHIT
           1        15  WIDTH INHIT
           1        16  THETA INHIT
           1        17  PHI INHIT
           1  INHIT                               11
  mtup=           2
  title:INHIT+OUTHIT
           2  OUTHIT                              18
           2        18  I OUTHIT
           2        19  J OUTHIT
           2        20  ENERGY OUTHIT
           2        21  TIME OUTHIT
           2        22  WIDTH OUTHIT
           2        23  THETA OUTHIT
```

```
         2          24  PHI OUTHIT
         2  OUTHIT                          18
mtup=           3
title:INHIT+OUTHIT+SHWR
         3  SHWR                            25
         3          25  ENERGY SHWR
         3  SHWR                            25

      1001
 INHIT                INHIT+OUTHIT+SHWR

         1  event ID
         2  Ld id
         3  Ld mass
         4  Ld Q
         5  Ld theta
         6  Ld phi
         7  Ld Etot
         8  Ld I
         9  Ld J
        10  sector
        11  I INHIT
        12  J INHIT
        13  ENERGY I
        14  TIME INH
        15  WIDTH IN
        16  THETA IN
        17  PHI INHI
        18  I OUTHIT
        19  J OUTHIT
        20  ENERGY O
        21  TIME OUT
        22  WIDTH OU
        23  THETA OU
        24  PHI OUTH
        25  ENERGY S
 opened Ntuple file "e_edge.rzdat" OK

 begin Ec_initialize_all...........................
version: July 1993 Ec3.1/NO_HOST/SIM/GRAPHICS
 ...Ec_init done...
 To exit enter "%"

 event ID:           1
 .........
 event ID:          67
ERROR: EcAcc_fill_Nth>EcAcc_request_Nth>EcDrv_request_Nth:Nth#1 of INNER HIT
      sector:6 illegal or nonexistent
ERROR: EcAcc_fill_Nth>EcAcc_request_Nth>EcDrv_request_Nth:Nth#1 of OUTER HIT
      sector:6 illegal or nonexistent
ERROR: EcAcc_fill_Nth>EcAcc_request_Nth>EcDrv_request_Nth:Nth#1 of SHOWER
      sector:6 illegal or nonexistent
 event ID:          68
 .........
 event ID:        1000
   END OF FILE found by EVFIO_GET_EVENT.
 .........
           1001 events processed
      CPU time/event:
      get         5.3273141E-04
      store       2.7805446E-03
      anal        0.1746950
      fill        4.3237675E-02
      reset       4.0957881E-03
      total       0.2627021
```