CLAS-NOTE-94-002

Ec 3.1.3

Forward Electromagnetic Calorimeter Reconstruction Software

K.B.Beard
beard@cebaf.gov

February 23, 1993

Ec 3.1.3 is the current release of the CLAS forward electromagnetic calorimeter (Ec) event reconstruction software. 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 host program; the **Ec.exe** program allows nearly all functions to be exercised one event at a time, while **Ec_engineG.exe** program calls Ec 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 CLAS-CODA 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, DEC-VMS, and HP-UX, but not yet under IBM-AIX. Future releases should use SwGen routines.

The Ec package compiles under HP-UX, DEC-Ultrix, and DEC-VMS and runs correctly under HP-UX and DEC-Ultrix; the required EvGen library routines are not supported under DEC-VMS. All details specific to the platform are hidden in the *KB_library* libraries. They consists of *KB_tricks* and *KB_special*.

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* (Figure 1). The **shortest** strip and its photomultiplier tube are 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.

Several coordinate systems are used by Ec (Figure 2). 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

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

position. The CLAS spherical system is the normal complement to the XYZ system.

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 on the inner face of the calorimeter 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 of projections parallel to the next edge in the sequence; it is defined such that, given the total length of each side L, $u/Lu + v/Lv + w/Lw = 2$ for all points. The point closest to the beam is the U corner; the U strips run in the J direction and are read out on the U edge. The measure along an edge is from its corner. The U edge ends at the V corner, and the V edge ends at the W corner, and the W edge ends at the U corner. This system is overdefined, but very convenient given the geometry 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 CLAS-CODA 2.0 event buffers. Commands may be issued directly by calling subroutines or by using the *pseudo_Ec* interface. **Ec.exe** and **Ec_engineG.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 *pseudo* interfaces at all; they are provided merely as a convenient way to communicate with the package.

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 except *pseudo_EcEngine.*

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 (`ECXXX>`) 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". Command files may be nested, and all *pseudo_Ec** routines support the "/HELP" command.

[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.

4 Output

The **Ec_engineG.exe** is an example of using the (EC ACCess) routines and putting the results into a 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 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_engineG.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], an interactive routine to project out 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, status, and meaning of the output results. The status should always be checked before the results are used.

5 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,

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

[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

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 (Figure 3). 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 Simulations

This release reads and writes CLAS-CODA 2.0 data files using the EvGen 2.0 package routines. The internal simulations are crude parameterizations intended 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 CLAS-CODA 2.0 format and counts photomultipliers the same way as Ec and the software standard. Both **CLAS315** and Ec get the location and dimensions of the calorimeters from the same include file **EcDba_basic.PAR**.

7 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; it assigns a reasonable energy to each pixel in a single

[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

pass, then forms hits from the pixels. Hits of the layers are combined to form showers using the same routine as the **Edge1** algorithm.

The **Edge1** 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. Corrections are made for attenuation of light along the strips. This routine refits the hits after corrections, but does not refit the peaks.

New algorithms may be easily added to the structure; two in planning are a variable-sized pixel Metropolis and and a continuation of Edge1 (Edge2) to support interative fitting of the peaks.

8 Acknowledgements

The author wishes to make clear that no Hampton University resources, time, or supplies were used during the development of Ec 3.1.3 from Ec 3.1.2, and that no encouragement, support, or responsibility of any kind should be attributed to the University. The author does wish to thank L.Dennis, T.Y.Tung, and M.Guidal for their support.

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*. The most generic routine is *Ec_recon_all*; it can by itself provide all Ec reconstruction given a CLAS-CODA event buffer.

Subroutine Ec_recon_all (idx, buffer, ptrs, OK, err)
Character(*)* setupfile
Integer CODACLASbuffer(*)
Character(*)* err
Logical OK

This subroutine will initialize Ec if required on the first call. Each time it is called, it will accept a CLAS-CODA 2.0 format event *buffer* with the index to the first word *idx* and filled lookup table *ptrs*, just as provided by *EvGen_get_event*. Then it will analyze everything it can, and report serious errors to *OK* and *err*. The results of the reconstruction are available via the *EcAcc* routines or via the *EcRsl* COMMONs.

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

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

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_status_report (data, done, sector, OK, err)
Logical data, done
Logical OK
Character(*)* err

For a specific *sector* and this routine returns whether the unpacking has been attempted (*done*), and whether there were any data unpacked during the attempt (*data*). Only if both *data* and *done* are true is there any reason to attempt reconstruction.

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; if the last value is also blank, the default method is used. Currently supported values are "Quick", "Fast", and "Edge1".

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 (other than the *EcRsl*) 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: EcEvu-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 EcRsl Access

An alternative to using *EcAcc* routines to recover results, the *EcRsl* COMMON blocks contain *EcDrv* (**Ec**Derived results) information in a different structure. This structure can be modified without affecting Ec as a whole. Unlike the *EcAcc* routines which check the validity of all arguments, the user must do so here.

D.1 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.

```
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

D.2 EcRsl Include Files

The *EcRsl* parameters are defined in **EcRsl_export.PAR**, and descriptions of each appear in **EcRsl_par_desc.CMN**. The status, whether there is valid information or not, appears in **EcRsl_status.CMN**. The actual reconstructed information is in **EcRsl_export.CMN**. The file **EcRsl_cross_reference.DTE** is used to define the mapping from the *EcDrv* information into *EcRsl* and is generally not needed by the user.

E 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 defined in **Ec_general.PAR** 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

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 *pseudo* 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; they were 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*
         /TUBEs                   tubes triggered

 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"
        "TUBE"

        "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 GeV
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:10."). 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_engineG Analysis Engine

A simple analysis engine, **Ec_engineG.exe**, demonstrates the use of the Ec package. It is designed to process CLAS-CODA 2.0 simulated data and output a CERNLIB Ntuple.

A single call

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

initializes all of the Ec package geometry and calibrations.

The pseudo_EcEngine routine is used to control the engine and give access to all other Ec interfaces. The EcAcc routines may be used to specify quantities to place in the output Ntuple. The appropriate Ntuple is booked, and the program enters a loop. The only proper exit from the engine is via the "/EXIT" option in the at the ECENGINE> prompt.

The program loops over events until it is halted by an error or reaching the number of events requested; then it returns to the ECENGINE> prompt. Each event the standard subrouine

```
   call EvGen_get_event(OK,err,in_idx,in_bffr,in_ptr,
  &                           CODA_in_hndl,ev_type,ev_id,procID)
```

provides a CLAS-CODA 2.0 format buffer and associated pointers, which are passed to:

```
call Ec_recon_all(in_idx,in_bffr,in_ptr,OK,err)
```

which stores the TDC and ADC values and performs all analysis.

The EcAcc routines are used to fill the Ntuple, and the Ntuple is then filled.

H.1 Ec_engineG Inputs

The Ec_engineG.exe code requires few instructions to run; for example the command:

```
Ec_engineG.exe < pi0.in > pi0_.out
```

runs the program "Ec_engineG.exe", taking input from "pi0.in" and sending output to "pi0_out". The file "pi0.in" is:

```
1                                                    !PAW workstation type 0=nothing
/usr/user4/beard/CLASsoft/GEANT/pizero.evt  !CLAS-CODA2.0 format event file
pi0_                                                 !rootname for Ntuple and logfile
!
!note: 1st 3 lines must be in sequence as shown, for rest order
!      not very important. KBB
!
/help                !list options for "pseudo_EcEngine" pseudoQ interface
!
/sectors:1::6        !consider sectors 1 to 6
!
/Nevents:100         !process only 100 events, then come back for instructions
/plot:ALL            !plot every event
!
Ec  Acc  @pi0_showers_only.ECACC !setup additional vectors for Ntuple
/maximum:2           !set maximum number of objects[vectors occurances]/event
!
!-set calibrations; not set quite right, but close
Ec  Cal  /-layer /-axis /-sector /unit:MeV /atten:99999. /Ech:3.960 /set:1::36
!
%continue on with program and begin processing- return to next line
!
/plot:NONE          !stop plotting
/Nevents:9999999    !continue for lots of events; stop at EOF
%continue on with program and begin processing- return to next line on EoF
!
Ec  Gra  /plot      !plot one last picture
/EXIT       !exit normally and close Ntuples properly
```

The file "pi0_showers_only.ECACC" is:

```
!
  /-char/-spec/-obj/-com                    !cancel everything old
  /com:ECDRV                                !specify common block
  /obj:SHOWER                               !specify object
  /char:ENERGY /char:X/char:Y/char:Z        !specify char.'s of interest
   SHWR/build                               !create a vector called "SHWR"
!
  /list                                     !list all vectors to screen
!
```

Multiple objects, or multiple occurances of an object, may be selected too. The order is arbitrary. In this example, all sectors were selected for output, with two object "SHWR" per sector. These Ntuples are of fixed size.

The following is a listing of a Ntuple written by **Ec_engineG.exe**; entries 1-23 come from the event descriptor block in the CLAS-CODA file, 23-107 from *EcAcc* vectors. Calculated quantities may be added to the list by modifying **Ec_engineG.F**. When the vector could not be filled, it was left empty. In this example, the beginning of a sector is denoted by a word, the number of occurances of an object to follow by *N_SHWR*, the 1st by *1_SHWR*, the id by *id.SHWR*, the requested characteristics by the form *char.obj* (trucated at 8 letters), *ENERGY.S*, *X.SHWR*, *Y.SHWR* and *Z.SHWR*.

```
PAW > ntup/print 1001

 ************************************************************
 * NTUPLE ID= 1001  ENTRIES=   2000   pi0_.rzdat
 ************************************************************
 *  Var numb  *   Name    *    Lower     *    Upper     *
 ************************************************************
 *      1     * N_Lds     *  .100000E+01 *  .100000E+01 *
 *      2     * Ld1_id    *  .111000E+03 *  .111000E+03 *
 *      3     * Ld1_stat  *  .000000E+00 *  .000000E+00 *
 *      4     * Ld1_Q     *  .000000E+00 *  .000000E+00 *
 *      5     * Ld1_x     *  .100000E-04 *  .100000E-04 *
 *      6     * Ld1_y     *  .100000E-04 *  .100000E-04 *
 *      7     * Ld1_z     *  .100000E-04 *  .100000E-04 *
```

```
*      8     * Ld1_Px    *  .258934E+00 *  .642754E+00 *
*      9     * Ld1_Py    *  .000000E+00 *  .000000E+00 *
*     10     * Ld1_Pz    *  .766073E+00 *  .965895E+00 *
*     11     * Ld1_Ptot  *  .150000E+01 *  .150000E+01 *
*     12     * Ld1_m     *  .000000E+00 *  .000000E+00 *
*     13     * Ld2_id    *  .000000E+00 *  .000000E+00 *
*     14     * Ld2_stat  *  .000000E+00 *  .000000E+00 *
*     15     * Ld2_Q     *  .000000E+00 *  .000000E+00 *
*     16     * Ld2_x     *  .000000E+00 *  .000000E+00 *
*     17     * Ld2_y     *  .000000E+00 *  .000000E+00 *
*     18     * Ld2_z     *  .000000E+00 *  .000000E+00 *
*     19     * Ld2_Px    *  .000000E+00 *  .000000E+00 *
*     20     * Ld2_Py    *  .000000E+00 *  .000000E+00 *
*     21     * Ld2_Pz    *  .000000E+00 *  .000000E+00 *
*     22     * Ld2_Ptot  *  .000000E+00 *  .000000E+00 *
*     23     * Ld2_m     *  .000000E+00 *  .000000E+00 *
*     24     * sector1   *  .100000E+01 *  .100000E+01 *
*     25     * N_SHWR    *  .000000E+00 *  .500000E+01 *
*     26     * 1_SHWR    *  .100000E+01 *  .100000E+01 *
*     27     * id.SHWR   *  .000000E+00 *  .100000E+01 *
*     28     * ENERGY.S  *  .000000E+00 *  .173282E+01 *
*     29     * X.SHWR    *  .000000E+00 *  .431867E+03 *
*     30     * Y.SHWR    * -.906139E+02 *  .121352E+03 *
*     31     * Z.SHWR    *  .000000E+00 *  .516038E+03 *
*     32     * 2_SHWR    *  .200000E+01 *  .200000E+01 *
*     33     * id.SHWR   *  .000000E+00 *  .200000E+01 *
*     34     * ENERGY.S  *  .000000E+00 *  .149083E+01 *
*     35     * X.SHWR    *  .000000E+00 *  .387655E+03 *
*     36     * Y.SHWR    * -.124599E+03 *  .110539E+03 *
*     37     * Z.SHWR    *  .000000E+00 *  .519122E+03 *
*     38     * sector2   *  .000000E+00 *  .200000E+01 *
*     39     * N_SHWR    *  .000000E+00 *  .100000E+01 *
.....................
*************************************************************
```

I Linking

The Ec package can be linked into a number of programs. The stand alone program **Ec.exe** exercises most functions; the shell **Ec_engineG.exe** will reconstruct CLAS-CODA files and store selected results to an Ntuple using the *EcAcc* routines, and **Ec_engineX.exe** uses the *EcRsl* commons. Ec can made part of a larger package; with or without graphics; and with or without pixels. By using dummy routines, Ec need not be modified if a host contains a CERNLIB PAWC common, but performs the necessary initializations of the memory, workstation, and graphics. The general Ec library includes both pixels and graphics; the PAWC common may be initialized by calling the appropriate routines.

Temporarily the files are on the CEBAF machines in the /usr/user4/beard/CLASsoft/EC3_1_3/* directories.

J Bugs

The HIGZ routines seem to have problems opening an Xwindow (workstation type 1) on a remote host, even if the environmental DISPLAY variable is set properly. If PAW is run first in that session and used to open a window, then exited, the HIGZ routines can open the window again. For example, on the Xwindow terminal ccxt6.cebaf.gov:

```
setenv DISPLAY ccxt6.cebaf.gov:0.0
paw
....
Workstation type (?=HELP) <CR>=1 : 1.ccxt6.cebaf.gov
....
PAW > quit
```

will work around the problem.

The PostScript figure files associated with this document do not print correctly on all PostScript printers. They were made using *pseudo_EcGra* and FrameMaker.

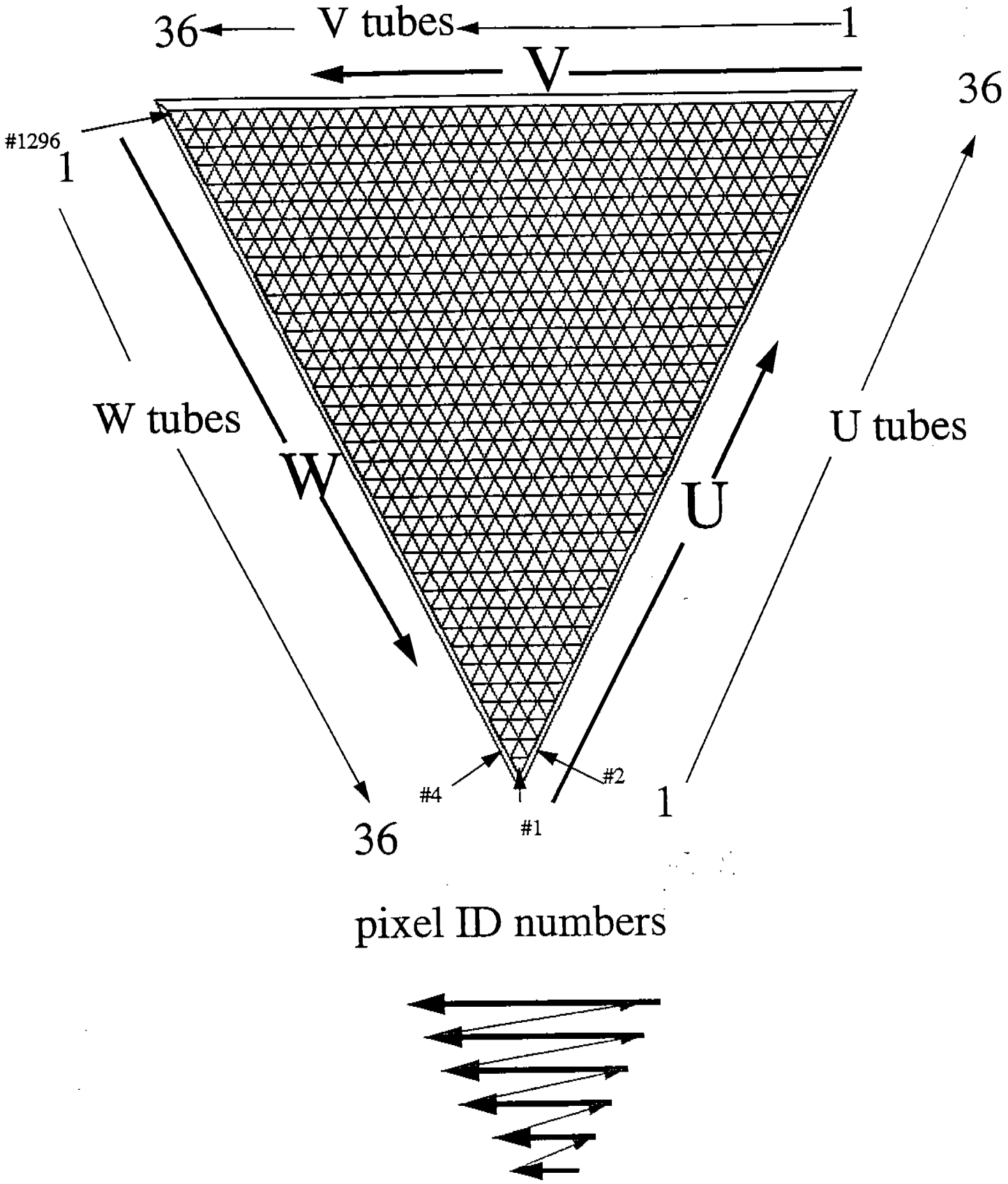

Figure 1. Ec Pixels

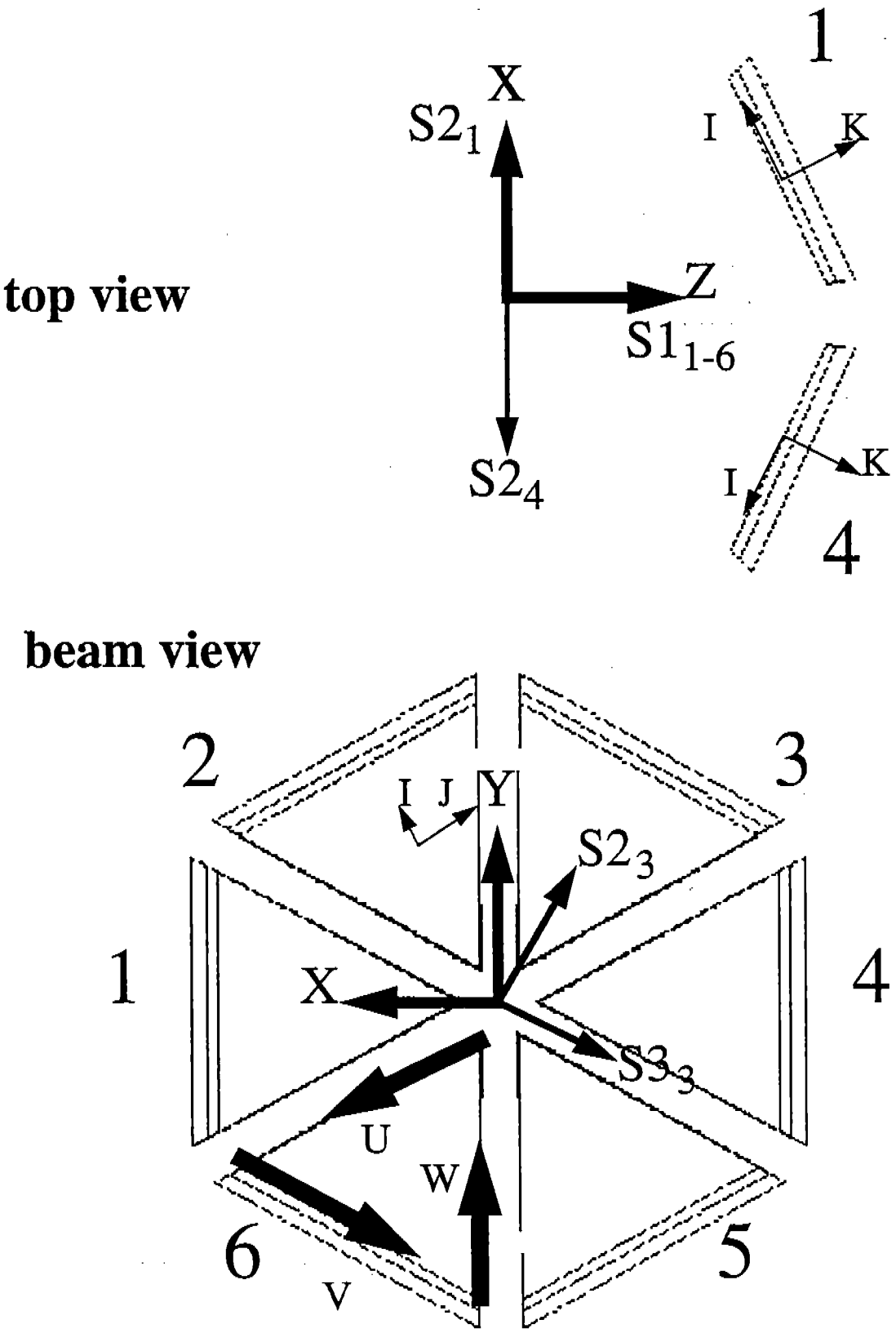

Figure 2. Ec Coordinate Systems

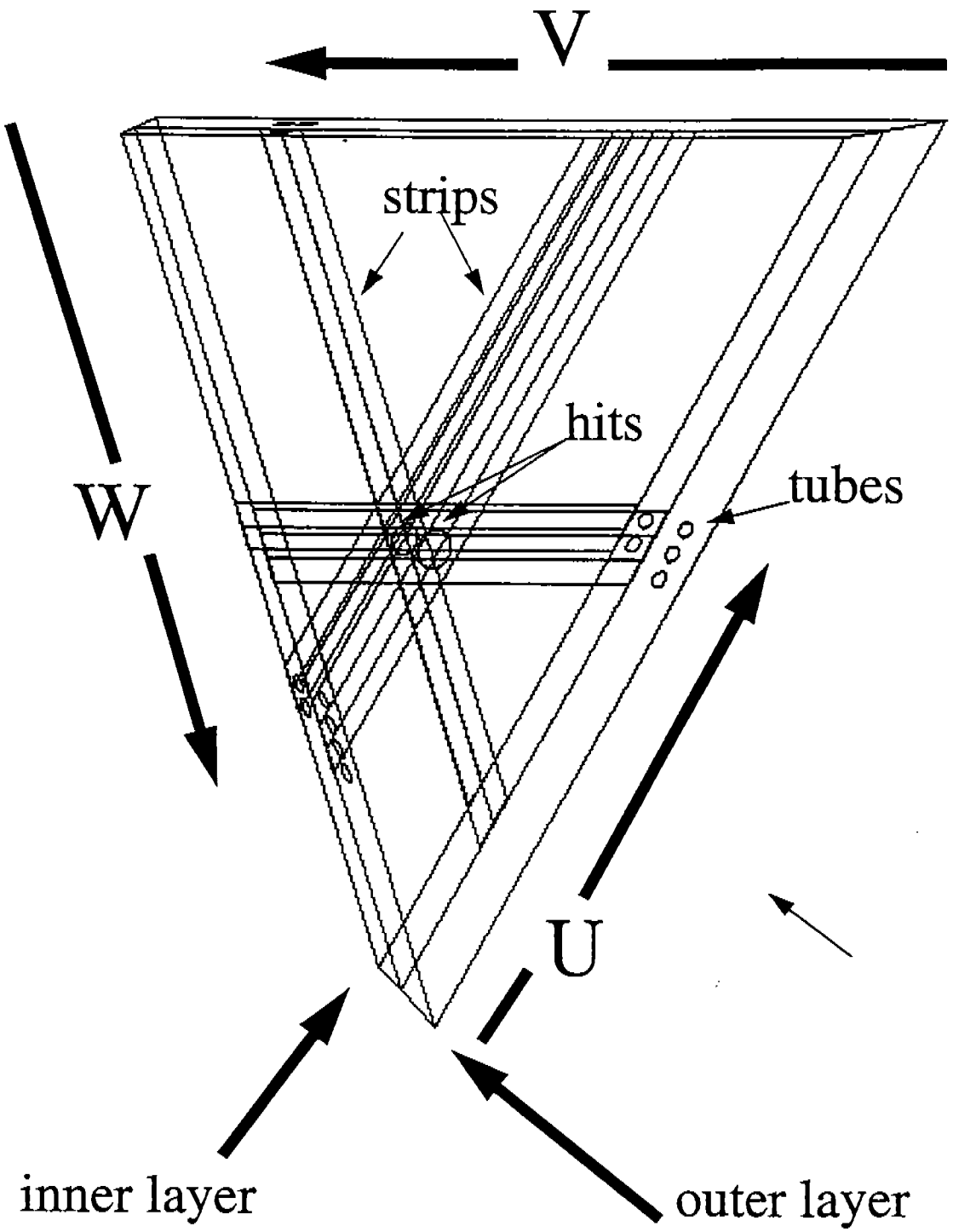

Figure 3. Ec Graphics