CLAS NOTE 94-004

Using the *ced* library

David P. Heddle
Christopher Newport University
heddle@clas01.cebaf.gov

Introduction

This document describes how you can link a FORTRAN code to the *ced* (cLAS eVENT dISPLAY) library: *libced.a*. The library provides the full functionality of the stand-alone version of *ced*: Multiple, adjustable, views of CLAS (both full-detector views and specialized package views) within subwindows which are individually controlled (zooming, scrolling, etc.) Events in the standard CLAS "CODA" format are overlaid on the detector components.

In addition, tracks can be drawn and analyzed shower counter data (in the form of pixel energies) can be displayed.

Obvious examples of codes that might benefit from linking to *libced.a* are simulation and analysis packages.You need only use a single call to the *ced* library to display events (provided that the events are in the CLAS format.)

Programs that have already linked to *ced* include the level-2 trigger simulation, CLAS GEANT (CLAS315), the CLAS analysis shell ("CSHELL"), EC (shower counter analysis) and the alternative drift chamber analysis package under development at UNH.

This report is organized as follows: In section 1, the hardware/software requirements are discussed. Where one can obtain *ced* is discussed in section 2. In section 3, the "big picture" strategy for using the *ced* library is discussed. An overview of the library routines is presented in section 4 followed by a more detailed discussion in section 5. Customizing the start-up appearance of *ced* is discussed in section 6. Finally, in section 7 a complete FORTRAN example is presented.

1) What are the requirements?

ced is supported on three platforms: ULTRIX (i.e. DECstations), AIX (IBM 6000 series RISC workstation) and HP-UX. A "renegade" version for DEC-Alpha (OSF) is available from the University of South Carolina. To run on any of these platforms, you need obtain the following:

1) *libced.a*, the compiled object library and/or *ced*, the stand-alone version

2) *clas.geometry*, the geometry database used by *ced*

3) *ced.help*, the help file used by *ced*'s on-line help facility (optional).

4) *clas3D.fmap*, the CLAS magnetic field table (optional). Note that you can also use the magnetic field table provided with the SDA release (usually called *bgrid.dat*). In *ced*, the magnetic field is only for display, it is not used to track particles.

5) *X Windows* and *Motif* libraries. If you have a unix workstation you already have the *X Windows* libraries and probably have the *Motif* libraries as well. Alternatively, you can link your program at CEBAF and ftp the executable back to your local system.

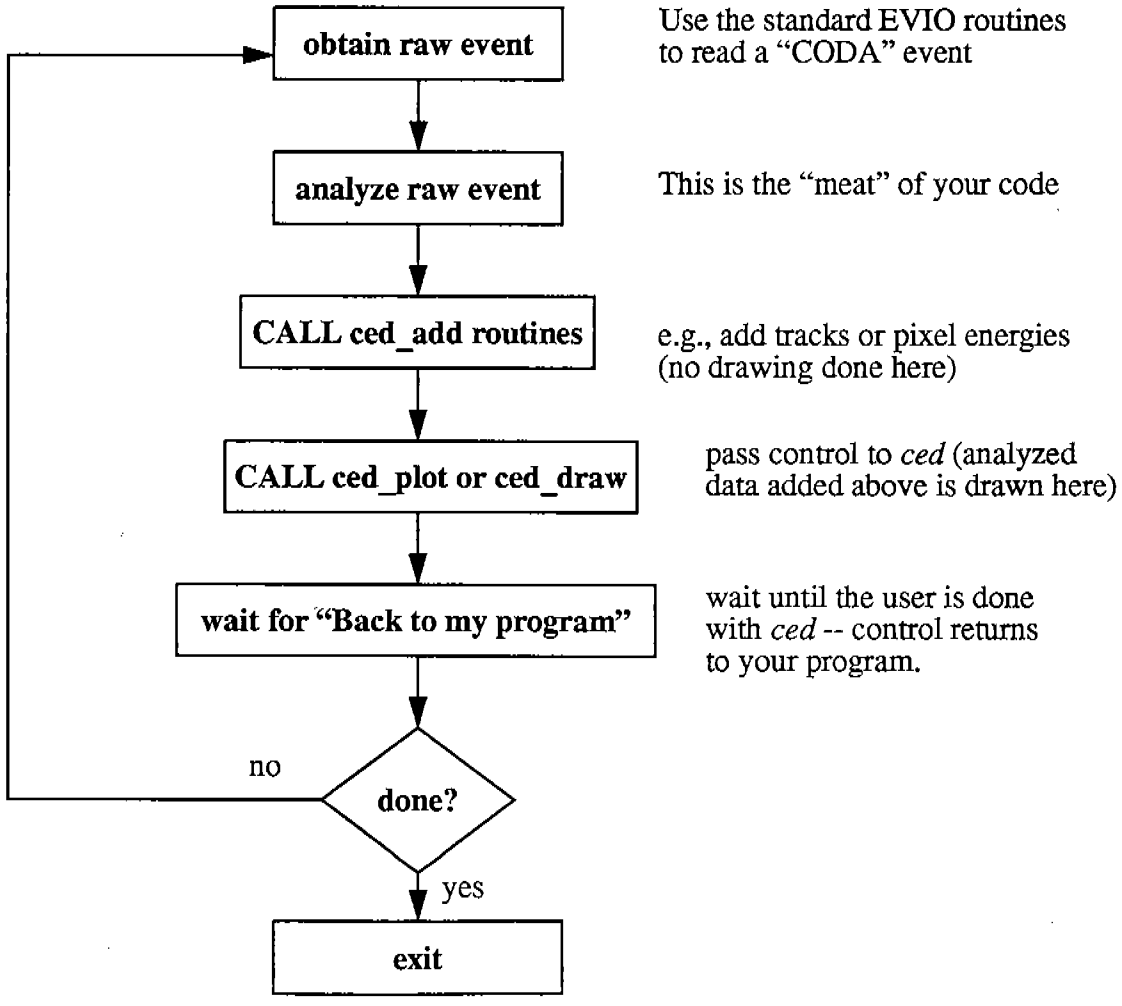
In addition, *ced* will only run on monitors with at least 256 colors (or shades of gray).

2) Where can I obtain these files?

For the source code, use the CLAS software, ftp site at FSU-SCRI. If you want to obtain the compiled files *ced* and *libced.a* directly (which I would offer up as the preferred approach), send an email request to *heddle@clas01.cebaf.gov.*

3) What is the big picture?

Most users will be linking to *ced* in order to display the results of some manner of analysis. The typical flow is thus: (details will be discussed later)



4 Using the *ced* library (implementation)

As discussed, obtain the library *libced.a* for your machine type (ULTRIX, HP-UX or AIX) or use existing public versions on clas01 (Ultrix), clas02 (HP-UX) or the FSU-SCRI ibm cluster. You will also need the geometry database: *clas.geometry*. You may also want to obtain the optional files *clas3D.fmap* and *ced.help*.

modify your "makefile" to link in *libced.a*

In version 1.4.x, the following calls are available:

(from FORTRAN)

```
CALL ced_add_pixels(n, sector, plane, u, v, w, energy)

CALL ced_add_track(pid, n, x, y, z)

CALL ced_draw

CALL ced_plot(ev_buffer)

CALL ced_refresh
```

Note that there is no explicit initialization. The library will initialize itself and pop-up the *ced* window the first time any of these routines is called.

Very important: The routines `ced_plot` and `ced_draw` (and no others) will cause control to be passed from your program to *ced*. Then you can zoom, print, etc. from the *ced* window. When you select "**Back to my program**" from the *ced* **Action** menu, control is passed back to your application. *ced*'s copy of all analyzed data that you added via `ced_add_pixels` or `ced_add_track` is deleted.

It is worth repeating in words what is shown on the flow chart on the previous page: *The general scheme is to get an event, analyze it, then call as many* `ced_add`*'s as you want. You will not "see" anything until you finally call* `ced_draw` *or* `ced_plot`. *When control returns to your application, all of* ced*'s analyzed data are deleted.*

5 Detailed explanations of the library functions.

```
ced_add_pixels(n, sector, plane, u, v, w, energy)
```

Arguments:

n	INTEGER*4	the number of pixel hits
sector	INTEGER*4 ARRAY(n)	for each hit 1..n, what sector it is in
plane	INTEGER*4 ARRAY(n)	for each hit 1..n, what plane: 0 for inner, 1 for outer

u	INTEGER*4 ARRAY(n)	for each hit 1..n, the ID of the coorsponding u strip
v	INTEGER*4 ARRAY(n)	for each hit 1..n, the ID of the coorsponding v strip
w	INTEGER*4 ARRAY(n)	for each hit 1..n, the ID of the coorsponding w strip
energy	INTEGER*4 ARRAY(n)	for each hit 1..n, the pixel energy in MeV

This routine provides a mechanism for passing reconstructed shower counter data to *ced*. The first argument `n` is an INTEGER*4 scalar. It specifies how many pixel "hits" are described in the following arrays. There is no limit on the value of `n`.

The remaining arguments are INTEGER*4 arrays. It is assumed they contain meaningful data in the indices `1..n`. *ced* will not alter the contents of the arrays. For hit `i`, `sector(i)` identifies the sector for the hit [1..6], `plane(i)` is either 0 (zero) for the inner shower counter or 1 for the outer. `u(i)`, `v(i)` and `w(i)` identify the u-strip, v-strip and w-strip [1..36] which comprise the pixel. `energy(i)` is the energy deposited on the pixel in MeV.

This routine does not initiate any drawing. The pixel data will not be displayed until a call to `ced_draw` or `ced_plot` is made. Control will *not* be given to *ced*. You can make multiple calls to `ced_add_pixels` prior to calling `ced_draw` or `ced_plot`.

After returning from `ced_draw` or `ced_plot`, all knowledge of the analyzed data is lost. This is done under the assumption that you will be moving on to the next event.

`ced_add_track(pid, n, x, y, z)`

pid	INTEGER*4	LUND id of the particle
n	INTEGER*4	the number of track points
x	REAL*4 ARRAY(n)	for each hit 1..n, what sector it is in
y	REAL*4 ARRAY(n)	for each hit 1..n, what plane: 0 for inner, 1 for outer
z	REAL*4 ARRAY(n)	for each hit 1..n, the ID of the corresponding u strip

This routine provides a mechanism for passing reconstructed tracks to *ced*. The first argument `pid` (INTEGER*4) is the LUND particle id (from the Particle Data Book). A relevant subset of these codes is provided in Table 1.

The next argument `n` is another INTEGER*4 scalar. It specifies how many track points are described in the following arrays. There is no limit on the value of `n`.

The remaining arguments are REAL*4 arrays. It is assumed they contain meaningful data in the indices `1..n`. *ced* will not alter the contents of the arrays. For hit `i`, `x(i)`, `y(i)` and `z(i)` are the x, y, and z points (cm) in the standard CLAS system.

This routine does not initiate any drawing. The track data will not be displayed until a call to `ced_draw` or `ced_plot` is made. Control will *not* be given to *ced*. You can make multiple calls to `ced_add_track` prior to calling `ced_draw` or `ced_plot`.

After returning from `ced_draw` or `ced_plot`, all knowledge of the track is lost. This is done under the assumption that you will be moving on to the next event.

particle	ID
γ	22
e^-	11
e^+	-11
ν_e	12
μ^+	-13
μ^-	13
π^0	111
π^+	211
π^-	-211
ρ^+ (770)	213
ρ^0 (770)	113
ω (1390)	223
η	221
n	2112
p	2212

Table 1: Particle IDs

```
ced_draw
```

(No arguments) This is equivalent to `ced_plot(0)`, as described below. It merely passes control to *ced*, which will display the detector and any analyzed data previously added via the `ced_add` routines. Thus one would use this when displaying only analyzed data without overlaying a raw event.

```
ced_plot(ev_buffer)
```

ev_buffer INTEGER*4 ARRAY() raw event buffer ("CODA" format)

This is the most important call in the library. It passes via the argument `ev_buffer` a raw event in the CLAS "CODA" format to *ced*, which will be displayed along with the detector and any ana-

lyzed data previously added via the `ced_add` routines.

This routine passes control to *ced*, which will display the detector and any analyzed data previously added via the `ced_add` routines along with the raw event passed in `ev_buffer`.

The call:

```
CALL ced_plot(0)
```

is equivalent to:

```
CALL ced_draw
```

`ced_refresh`

(No arguments) This routine causes *ced* to redraw its window. It should only be needed if, while your program has control, you cause the *ced* window to be occluded and re-exposed. Since the *ced* window is only sure to be "current" what it has control, you probably won't care that its contents are trashed while you are performing analysis. When *ced* has control, it will handle occlude-expose sequences automatically.

6 Customizing the *ced* window

Like any other X-Windows based program, *ced* is customized through a resource file. When using the library version of *ced*, the appropriate resource file is named *Lced*.

You may copy an existing version of *Lced* (ascii text) from clas01 in /clas01/usr/lib/X11 or from clas02 in /clas02/usr/users/heddle/app-defaults. You can then modify it to reflect the desired customizations, such as the size and position of the *ced* window at start-up or the background color.

When starting, the system will look for *Lced* in standard places such as /usr/lib/X11/app-defaults. Since you probably lack the privileges necessary to put files in those places, it is necessary to direct the system to look someplace else. This is done through the XAPPLRESDIR environment variable. The simplest procedure is:

1) From your home directory, create a subdirectory named app-defaults. Place your copy of *Lced* in that directory.

2) In your .login file, add the line:

   setenv XAPPLRESDIR [path to your home directory]/app-defaults

   e.g., on clas02 my .login contains the line:

   setenv XAPPLRESDIR /clas02/usr/users/heddle/app-defaults

3) Either logout and re-login or "source" your .login:

$source .login

Now the system should use your private copy of *Lced.*

Here is the present "default" version of Lced. I will assume that making modifications is self-explanatory.

```
!
! Defaults for Lced
!
*allowShellResize:true
*borderWidth:0
*highlightThickness:2
*traversalOn:true

#ifdef COLOR
 *background: skyblue
 *foreground:black

 *XmToggleButtonGadget*selectColor:Orange
#endif

*XmMessageBox.labelFontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*
*XmMessageBox.buttonFontList:-*-helvetica-bold-r-normal--14-*-*-*-*-*-*-*

*XmPushButtonGadget.fontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*
*XmToggleButtonGadget.fontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*
*XmCascadeButton.fontList:-*-helvetica-bold-r-normal--17-*-*-*-*-*-*-*
*XmCascadeButtonGadget.fontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*

*XmFileSelectionBox.XmText.fontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*
*XmFileSelectionBox.buttonFontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*
*XmFileSelectionBox.labelFontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*
*XmFileSelectionBox.textFontList:-*-helvetica-medium-r-normal--14-*-*-*-*-*-*-*

*Lced.x:60
*Lced.y:37
*Lced.width:1126
*Lced.height:898

*colorpanel*numColumns: 32
*colorpanel*adjustLast: False
*colorpanel*packing: pack_column
```

Background Color

Window placement and size

7 Example: A simple event-fetcher

As an example, included in this document is a simple event-fetching program written by Larry

Dennis. The program will read CLAS events from the file *evt_input.evt* one-at-a-time until the end-of-file. Within this loop, a call to the *ced* library is made to display the event.

Note that this program is an ideal "skeleton" for developing an analysis code, since it already handles all the i/o for the standard CLAS format. One would merely, in the same manner as the call to `ced_plot`, "intercept" the event and pass it to your analysis routines. One might then imagine removing the call to `ced_plot` from the loop, and instead placing its invocation under the control of the user (i.e. you probably don't want to see every event.)

What follows is the makefile used to generate this "skeleton", followed by a complete listing. One can obtain these files from clas01 in the directory /clas01/usr/local/ced/src.

Sample makefile (available from /clas01/usr/local/ced/src/makefile)

```
# Creating evt_test executable module
#
f77 = f77 -g -cpp -G 1600
#
# Define the base directories for SDA_TEST_IO
#
LCD_ROOT = /clas01/usr/users/dennis_l
#
# Define symbols for filenames.
#
# Use development versions of evgen and evfio routines.
#
CODA = $(LCD_ROOT)/newcoda
EVIO = $(LCD_ROOT)/newcoda
#
# Locate the ced library,
#
CED = /clas01/usr/local/lib
XLIB = -lX11
XTLIB = -lXt
XMLIB = -lXm

MAIN = evt.exe
#
# Define library linking routines.
#
IOLIB = -L$(CODA) -levgen -L$(EVIO) -levfio
CEDLIB = -L$(CED) -lced
#
# Define the object files needed for the test package.
#
ST = evt
#
OBJS = $(ST).o

$(MAIN): makefile $(OBJS)
$(f77) -v -o $(MAIN) $(OBJS) \\p$(IOLIB) $(CEDLIB) $(XMLIB) $(XTLIB) $(XLIB)
strip $(MAIN)

EVGEN.PAR: $(CODA)/EVGEN.PAR
cp $(CODA)/EVGEN.PAR .

$(ST).o: $(ST).f EVGEN.PAR
$(f77) -c -o $(ST).o $(ST).f
```

Sample fortran "driver"(available from /clas01/usr/local/ced/src//evt.f)

```
C=======================================================================
C LEVEL 1, EVT_MAIN
C-----------------------------------------------------------------------
 PROGRAM EVT_MAIN
C-----------------------------------------------------------------------
C-
C- Purpose and Methods :
C-
C- Libraries required :
C- Inputs :
C- Outputs :
C- Controls:
C-
C- Created 24-May-1993 Larry Dennis
C-
C-----------------------------------------------------------------------
 IMPLICIT NONE
C-----------------------------------------------------------------------
C
C-- Initialisation phase
 CALL evt_test_init
C
C
C-- Processing phase
 CALL evt_test_go
C
C-- Termination phase
 CALL evt_test_last
C
 STOP
 END
C=======================================================================
 SUBROUTINE evt_test_go
C-----------------------------------------------------------------------
C-
C- Purpose and Methods : Steering routine for event processing
C- This test version select dc superlayer 3,
C- the scintillators and the outer
C- electromagnetic calorimeter and writes them
C- out. It uses the extract and build routines
C- to do this, so it tests those routines.
C-
C- Inputs :
C- Outputs :
C- Controls:
C-
C- Created 24-May-1993 Larry Dennis
C-
C-
C- Called by EVT_TEST
C-
C-----------------------------------------------------------------------
 IMPLICIT NONE

 INCLUDE 'EVGEN.PAR'
```

```
      LOGICAL OK
      CHARACTER*80 error

      INTEGER in_handle, out_handle
      COMMON /Chandle/ in_handle, out_handle

      INTEGER in_buff, in_ptrs
      INTEGER out_buff, out_ptrs
      DIMENSION in_buff( 8192 ), out_buff(8192)
      DIMENSION in_ptrs( 0:EVGEN_MAX_SEC, 0:EVGEN_MAX_PKG,
     & 0:EVGEN_MAX_SUB )

      DIMENSION out_ptrs( 0:EVGEN_MAX_SEC, 0:EVGEN_MAX_PKG,
     & 0:EVGEN_MAX_SUB )

      INTEGER NSEC
      PARAMETER( NSEC = 6 )
      INTEGER dcsda_hits, scsda_hits, ecsda_hits, ccsda_hits
      INTEGER dcsda_digi, scsda_digi, ecsda_digi, ccsda_digi
C
C xxsda_hits contains the number of hits in a subpackage.
C
      DIMENSION dcsda_hits( 36, NSEC )
      DIMENSION scsda_hits( 1, NSEC )
      DIMENSION ccsda_hits( 2, NSEC )
      DIMENSION ecsda_hits( 6, NSEC )
C
C xxsda_digi contains the id, tdc, adc, etc of each component.
C
      DIMENSION dcsda_digi( 3, 40, 36, NSEC )
      DIMENSION scsda_digi( 5, 10, 1, NSEC )
      DIMENSION ecsda_digi( 3, 36, 6, NSEC )
      DIMENSION ccsda_digi( 3, 18, 2, NSEC )

      INTEGER dcdesc(8), scdesc(8), ecdesc(8), ccdesc(8)

      INTEGER program_id, proc_id, evt_type, evt_id, ptr, optr

      Character*80 ostring
      INTEGER event_cnt, sec_id, pkg_id, sub_id, unit
      CHARACTER*80 string

      INTEGER*4 nseg, iseg, jseg, segarray(3,64)

      DATA event_cnt / 1 /
      DATA unit / 6 /
      DATA string / 'Test Routine Full Event Display' /
C
C Fill descriptions arrays with: event_type, sec_id, pkg_id,
C pkg_count, nprms, ncmps, nsubs, nsect
C
      DATA dcdesc / EVGEN_PCK_EVT, 0, EVGEN_DC_ID, EVGEN_DC_PKG_COUNT,
     & 3, 40, 36, NSEC /
      DATA scdesc / EVGEN_PCK_EVT, 0, EVGEN_SC_ID, EVGEN_SC_PKG_COUNT,
     & 5, 10, 1, NSEC /
      DATA ecdesc / EVGEN_PCK_EVT, 0, EVGEN_EC_ID, EVGEN_EC_PKG_COUNT,
     & 3, 36, 6, NSEC /
      DATA ccdesc / EVGEN_PCK_EVT, 0, EVGEN_CC_ID, EVGEN_CC_PKG_COUNT,
     & 3, 18, 2, NSEC /
```

```
C
C-------------------------------------------------------------------------
C
 program_id = 0
c
c Begin loop over the events.
c
 10 CONTINUE
 evt_id = 0
 proc_id = 0
 evt_type = EVGEN_PCK_EVT
 ptr = 1
 call evgen_get_event( OK, error, ptr, in_buff, in_ptrs,
 & in_handle, evt_type, evt_id, proc_id )
 IF( OK ) THEN
c
c Ask ced to plot the event
c
C-- Start ced access

      print*, 'Calling ced...'
      CALL ced_plot(in_buff)

      print*, 'Back from ced.

C- End of ced access'
 DO sec_id = 1, NSEC
c
c Set the sector id. Note that order of extraction doesn't
c matter.
c
 dcdesc(2) = sec_id
 ecdesc(2) = sec_id
 scdesc(2) = sec_id
 ccdesc(2) = sec_id
 call evgen_extract_all( OK, error, in_buff, scdesc,
 & scsda_hits, scsda_digi, in_ptrs )
 call evgen_extract_all( OK, error, in_buff, ecdesc,
 & ecsda_hits, ecsda_digi, in_ptrs )
 call evgen_extract_all( OK, error, in_buff, dcdesc,
 & dcsda_hits, dcsda_digi, in_ptrs )
 call evgen_extract_all( OK, error, in_buff, ccdesc,
 & ccsda_hits, ccsda_digi, in_ptrs )
 END DO
c call your_analysis(dcsda_hits, dcsda_digi)
 GO TO 10
 ELSE
 write(6,*) ' Error encountered on input '
 write(6,'(a80)') error
 END IF
 999 RETURN
 END
C==========================================================================
 SUBROUTINE EVT_TEST_INIT
C-------------------------------------------------------------------------
C-
C- Purpose and Methods : Initialization of EVT_TEST program.
C-
```

only place where *ced* is called

```
C- Inputs :
C- Outputs :
C- Controls:
C-
C- Created 24-May-1993 Larry Dennis
C-
C-
C- Called by EVT_MAIN
C-
C----------------------------------------------------------------------
 IMPLICIT NONE
C----------------------------------------------------------------------
C
 INTEGER in_handle, out_handle
 COMMON/Chandle/ in_handle, out_handle
 character*80 inevent, outevent, error
 LOGICAL OK
 character*8 flags
C
C Open Files for input and output of raw events.
C
 WRITE(6,*) ' Open input file.'
 inevent = 'evt_input.evt'
 flags = 'r'
 OK = .TRUE.
 CALL evgen_open(OK, error, inevent, flags, in_handle )
 IF( .NOT. OK ) THEN
 write(6,*) ' Return from evgen_open '
 write(6,'(a)' ) error
 STOP
 ELSE
 WRITE(6,*) ' Open output file.'
 outevent = 'evt_output.evt'
 flags = 'w'
 CALL evgen_open(OK, error, outevent, flags, out_handle )
 IF( .NOT. OK ) THEN
 write(6,*) ' Return from evgen_open '
 write(6,'(a)' ) error
 END IF
 END IF
 999 RETURN
 END
C======================================================================
 SUBROUTINE EVT_TEST_LAST
C----------------------------------------------------------------------
C-
C- Purpose and Methods : Shut down of EVT_TEST program.
C-
C- Inputs :
C- Outputs :
C- Controls:
C-
C- Created 24-May-1993 Larry Dennis
C-
C-
C- Called by EVT_MAIN
C-
C----------------------------------------------------------------------
 IMPLICIT NONE
```

```
C-----------------------------------------------------------------------
C
 INTEGER in_handle, out_handle
 COMMON/Chandle/ in_handle, out_handle
 LOGICAL OK
 CHARACTER * 80 error
C
C Close files for input and output of raw events.
C
 WRITE(6,*) ' Close input file.'
 CALL evgen_close( OK, error, out_handle )
 IF( .NOT. OK ) THEN
 write(6,*) ' Error on output close '
 write(6,'(a)' ) error
 ELSE
 WRITE(6,*) ' Close output file.'
 CALL evgen_close( OK, error, in_handle )
 IF( .NOT. OK ) THEN
 write(6,*) ' Error on input close '
 write(6,'(a)' ) error
 END IF
 END IF
 999 RETURN
 END
```