CLAS-NOTE-94-023

SwGus 1.1.2

SoftWare group General Utility Software

K.B.Beard

November 28, 1994

The CLAS software project programmers face common obstacles and problems. This is a set of solutions following the CLAS software standards developed for the Ec package and general use.

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 forward electromagnetic calorimeter (Ec) package [1] [2] was conceived to link to some (as of then) unspecified host code; many temporary solutions to the problems of input, output, command and control were implemented. These were gathered together under the standard name; many were taken from the author's personal libraries.

2 Coding Comments

Since debugging will be such an important part of the development effort, considerable effort was directed to simplifying the task as much as possible. Most routines have a parameter **here** which is defined to contain the name of the subroutine; this is used to allow the FORTRAN routines to generate useful self-documentation.

Most SwGus routines have been written to conform to the CLAS software standards [3]; older code was modified to comply. The SwGus package compiles and runs correctly under DEC-VMS, DEC-Ultrix, and HP-UX.

2.1 Interactive Interface

The interactive interface is one taken from the pseudoQ package [4]; it allows deeply nested command files and interfaces and is generally very similar to

[1] Ec 3.2.1, Forward Electromagnetic Calorimeter Reconstruction Software, CLAS-NOTE-94-024

[2] Ec 3.1.3, Forward Electromagnetic Calorimeter Reconstruction Software, CLAS-NOTE-94-002

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

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

the LAMPF Q system interface [5].

2.2 Error Messages

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 unless a insurmountable problem occurs, such as a file not being found or encountering an uninitialized package.

2.3 Data Format

The SwGus package supports the BOS 0 [6] format; for backward compatablility CLAS-CODA 2.0 format [7] is also included. It is assumed that all inter-package communication is via BOS banks in the common contained within the **SwGus_BOS.CMN** INCLUDE file.

3 Files

The library and sources should shortly be found in the standard CLAS directory and the anonymous FTP site. The only INCLUDE files intended for use outside of SwGus are **SwGus_BOS.CMN**, containing the main BOS common, and **SwGus_BOS.DEC**, a list of declarations of BOS functions.

A brief description of each routine is in the Appendix.

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

[6] CLAS Event Format with BOS, Version 1.00, CLAS-NOTE 94-012

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

A CERNLIB Related Routines

The CERNLIB library tends to be changed periodically, so standardizing the calls to it will simplify upgrades.

Subroutine SwGra_initialize_workstation (wrkstn_type, OK, err)
Integer wrkstn_type
Character(*)* err
Logical OK

Given a CERNLIB workstation of type *wrkstn_type*, this routine attempts to initialize CERNLIB working space by calling **MZPAW**, **HLIMIT**; if *wrkstn_type* is non-zero, it initializes HIGZ graphics using **IGINIT**, **IGSSE**, and **IGSA**.

Subroutine SwGus_initialize_workstation (wrkstn_desc, OK, err)
Character(*)* wrkstn_desc, err
Logical OK

Given a CERNLIB workstation described by *wrkstn_desc* of the form **N** .or **N.node.internet.address**, where **N** is an integer workstation type, this routine attempts extract the workstation type and call **SwGra_initialize_workstation**. It is included for future remote network access capabilities.

Subroutine SwGus_list_workstation_choices (IO)
Integer IO

Writes a list of available CERNLIB workstation types to (already opened) FORTRAN channel *IO*.

Subroutine SwGus_prepare_workstation (wrk, OK, err)
Integer wrk
Logical OK
Character(*)* err

Performs any additional, platform dependent initializations required after **SwGra_initialize_workstation**. It currently does nothing.

B BOS 0 Related Routines

The following routines are all based on the BOS manual and BOS 0 format.

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

Initializes the BOS COMMON; must be done before any subsequent use of BOS routines; prevents reinitializing the COMMON.

Subroutine SwGus_BOS_sim_tag (first, last, Nmax, tag, OK, err)
Integer first, last, Nmax
Character(*)* tag(*), err
Logical OK

Fills the array *tag*, beginning with index *first*, with appropriate Ntuple tag names to contain the BOS 0 event description bank **MCIN**. The number of particles is limited by *Nmax*, *last* is the index of the last *tag* used. The array *tag* is for use in creating an Ntuple via the **HBOOKN** call.

Subroutine SwGus_BOS_sim_tuple (first, last, Nmax, tuple, OK, err)
Integer first, last, Nmax
Real tuple(*)
Logical OK
Character(*)* err

The event description is extracted from BOS 0 bank **MCIN** and put them into the Ntuple array *tuple* beginning with index *first* before the **HFN** call. The *last* index used is returned.

Subroutine SwGus_BOS_comment (bank_name, cmmnd, comment)
Character(*)* bank_name, cmmnd, comment

Creates or reads a BOS documentation bank containing a useful description of a general BOS bank. The documentation and the bank it describes have the same name, but are stored in different COMMONs. The command *cmmnd* tells this routine whether to copy the *comment* into or from the (auxillary) BOS bank *bank_name*. If (*cmmnd*= "<" or "<=" it copies the *comment* into the bank; if (*cmmnd*= " ", ">", or "=>" it copies the *comment* from the bank. The *comment* may be of any length.

C Character String Manipulation Routines

These are used to simplify input and output and avoid wasting effort on case sensitivity, spacing, etc.

Subroutine SwGus_add_path (OK, where, err)
Logical OK
Character(*)* where, err

Upon failure (*OK*=.FALSE.), the name of the current routine *where* is prepended to the beginning of expression *err* with an appropriate delimiter, if necessary. This routine is used when passing error messages up to a higher routine.

Subroutine SwGus_ASCII_I4_list (N, V, Cx, lenCx)
Integer N, V(*), lenCx
Character(*)* Cx

Given a list V of N integers, produces a neat list in Cx with nonblank length $lenCx$ using PseudoQ notation; i.e., "1, 2, 3, 4, 9" becomes "1::4:9".

Subroutine SwGus_cleanup (note, nonblank_length)
Character(*)* note
Integer nonblank_length

Given a string *note*, removes redundant blanks, nulls, and tabs, left justifies, and returns the index of the last non-blank character.

Subroutine SwGus_default_extension (file, ext)
Character(*)* file, ext

Given a filename *file*, will place the extension *ext* on the end if the filename does not already have an extension. The "." will be automatically added if required.

Subroutine SwGus_get_I4_values (string, n, values, ok)
Character(*)* string
Integer n, values(*)
Logical OK

Given a *string*, attempts to read a list of integer *values*; returns success or failure (*ok*) and the number of values found *n*.

Subroutine SwGus_get_R4_values (string, n, values, ok)
Character(*)* string
Integer n
Real values(*)
Logical OK

Similar to **SwGus_get_I4_values**, but returns real values. Integers are converted into floating values.

Integer Function SwGus_important_length (string)
Character(*)* string

Given *string*, this routine removes leading blanks and returns the index of the last nonblank character.

Logical Function SwGus_match (test, pattern)
Character(*)* test, pattern

This routine returns .TRUE. only if *test* matches *pattern*. DCL-like rules are used; the *test* must contain at least up to the *pattern*'s first "*", any characters beyond that must also match but need not be present. Both arguments are case insensitive. For example, *pattern*=**TE*STS** is matched by *test*=**Test**, but not *test*=**testing**.

Subroutine SwGus_mss (pat, wildI, Ival, wildR, Rval, fmt, note)
Character(*)* pat, wildI, wildR, fmt, note
Integer Ival(*)
Real Rval(*)

Copies the pattern *pat* into *note*; first seeks the integer wildcard character *wildI* and replaces each occurance with the next value of *Ival*, then repeats for *wildR* and value of *Rval*. If *fmt* is nonblank, it replaces the default FORTRAN format used. Blank values of *wildI* and *wildR* are ignored.

Subroutine SwGus_NO_blanks (string)
Character(*)* string

Removes all tabs and nulls; moves all blanks to end of *string*.

Subroutine SwGus_NO_comments (string, delimiters)
Character(*)* string, delimiters

Seeks any of the characters in *delimiters* (terminated by a blank) within *string*; if found, replaces that character onward with blanks.

Subroutine SwGus_NO_leading_blanks (string)
Character(*)* string

Moves all leading blanks within *string* to the end.

Subroutine SwGus_NO_nulls (line)
Character(*)* line

Replace all null characters within *line* with blanks.

Subroutine SwGus_NO_tabs (line)
Character(*)* line

Replace all tab characters within *line* with blanks.

Subroutine SwGus_only_one_blank (string)
Character(*)* string

Moves all redundant blanks (second or higher in a row) to the end of *string*. Removes all leading blanks.

Subroutine SwGus_real_data_row (string)
Character(*)* string

Attempts to "clean up" *string* so a FORTRAN READ(*string*,*) can understand it; packs to end with "," characters.

Subroutine SwGus_ShiftAll (InPut, OUTPUT)
Character(*)* InPut, OUTPUT

Copies the string *Input* to *OUTPUT*, removes leading blanks, shifts all characters to upper case.

Subroutine SwGus_ShiftLen (old, NEW, nonblank)
Character(*)* old, NEW
Integer nonblank

Similar to **SwGus_ShiftAll**, but also returns the index *nonblank* of the last nonblank character.

Subroutine SwGus_squeeze (line, nonblank)
Character(*)* line
Integer nonblank

Removes all nulls, tabs, and blanks; returns the number of nonblank characters *nonblank* in *line*.

Subroutine SwGus_substring (string, old, new)
Character(*)* string, old, new

Seeks the character pattern *old* within *string*; replaces all occurances with *new*.

Subroutine SwGus_UP_shift (string)
Character(*)* string

Shifts any lower case characters within *string* to upper case.

D General Utility Routines

The following routines are used to reorder lists for convenience.

Subroutine SwGus_int_sort (N, list, idx)
Integer N, list(*), idx(*)

Given a *list* of length *N*, fills *idx* with the relative location within *list* of the smallest to the largest.

Subroutine SwGus_sort (N, size, threshold, idx)
Integer N, idx(*)
Real size(*), threshold

Given a list *size* of length *N*, fills *idx* with the relative location within the list of the largest to the smallest; *N* becomes the number of values above *threshold*.

E FORTRAN I/O Routines

Often, the I/O operations prove the most troublesome. These routines attempt to simply the problem.

Subroutine SwGus_IO_control (who, IO, command, OK, err)
Character(*)* who, command, err
Integer IO
Logical OK

This routine is intended to prevent two programmers from trying to use the same FORTRAN IO channels at the same time, and to keep track of who is using what channel. The calling subroutine (and any comments) is *who*, *IO* is the FORTRAN IO channel number in question, *command* tells the routine what to attempt to do, while *OK* and *error* report success or failure, and an explanation of the failure, if any. The *command* is case insensitive. To request any unused channel, *command* = "?...", "A", or "ASK". To request a specific channel, *command* = "=...", "R", or "REQUEST". To release a channel for other use, *command* = "!...", "F", "FREE", or "FINISHED". To provide a status report to channel *IO* of who is using what channel, *command* = "#...", "S", or "STATUS"; if *IO* is not opened or 0, the output is directed to *IO*=6. The "..." denodes optional comments.

Subroutine SwGus_NewFile (who, filename, IOhndle, OK, err)
Character(*)* who, filename, err
Integer IOhndle
Logical OK

When user routine *who* requests that a new text file *filename* be opened, this routine returns the *IOhndle* assigned (by **SwGus_IO_control**) and success or failure and any explanation in *OK, err*.

Subroutine SwGus_OldFile (who, filename, IOhndle, OK, err)
Character(*)* who, filename, err
Integer IOhndle
Logical OK

When user routine *who* requests that an existing text file *filename* be opened, this routine returns the *IOhndle* assigned (by **SwGus_IO_control**) and success or failure and any explanation in *OK, err*.

Subroutine SwGus_rep_err (OK, note)
Logical OK
Character(*)* note

When *OK*=.FALSE., this routine calls **SwGus_wrap_note(**6, 'ERROR:'//*note)*. It is the standard utility for reporting and logging errors.

Subroutine SwGus_wrap_note (IOchannel, note)
Character(*)* note
Integer IOchannel

This routine writes *note* out to (already opened) FORTRAN IO channel *IOchannel*; it cleans the message up by removing extraneous blanks, tabs, and nulls and provides nice wrapping and indenting of subsequent lines.

F PseudoQ Interface Routines

These routines are used in the *pseudo_** interfaces. Advantages of these routines are that both interfaces and command files may be nested to great depth and a terse, simple, and unambiguous syntax. All pseudoQ commands of the form:

```
Field1/op1 /op2/op3:n FIELD2/op4:string /op5:i:j:k
```

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 "&". Often the fields usually provide information for the options. Commands are not case sensitive but the case of file names is preserved for UNIX compatability.

If desired, in an interface another interface routine may be specified by the first field. If only the field is specified, the corresponding subroutine (pseudo_YYY) 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" and may contain other command files, also prefixed with an "@"; upon reaching the end of a nested file, control is returned to the calling file.

Subroutine SwGus_parse_command (inline, N_name, name, N_option, option)
Character(*)* inline, name(*), option(*)
Integer N_name, N_option

Given input line *inline*, this routine uses PseudoQ syntax to extract the number *N_name* of (non-option) words *name* and the number *N_option* of options *option*. Options are words prefaced with "/".

Subroutine SwGus_parse_i4_Option (option, opt, kind, value, string)
Character(*)* option, opt, string
Integer kind, value(*)

This routine parses PseudoQ options. All *option*s are of the form *opt:string*; if *string* is blank *kind*=0, if it cannot be interpreted as a list of integers *kind*=-1, and if it can *kind* contains the number of integer *value*s recovered.

Subroutine SwGus_Qcommand (prmpt, inst, Nfld, fld, Nopt, opt, key, ret, kill)
Character(*)* prmpt, inst, fld(*), opt(*)
Integer Nfld, Nopt
Logical key, ret, kill

This routine is the heart of the PseudoQ interface; the package using it is identified by *prmpt*, which is used for the user prompt. If the instruction *inst* is blank, the routine assumes interactive mode, provides the appropriate prompt, and awaits input. If *inst* is nonblank, that one instruction is parsed and the routine returns. The *key* reports whether the routine is expecting input from the keyboard, *ret* indicates that the calling routine is expected to exit after processing the *fld*s and *opt*s, and *kill* indicates that an immediate return should be made. The number *Nfld* of fields *fld* are the words not beginning with "/"; the number *Nopt* of options *opt* are those which are preceeded by a "/". Command files are indicated by a "@" preface and may be nested many layers deep. This routine can be used by many packages simultaneously; all IO control is maintained internally and based on **SwGus_IO_control**.

```
SUBROUTINE pseudo_Xxx(firstline,OK)
....
character*(*) firstline
logical OK
....
LOGICAL SwGus_match                    !FUNCTION
....
line= firstline
DO WHILE (.NOT.return)
  call SwGus_Qcommand('Xxx',line,Nfld,fld,Nopt,opt,key,ret,kill) !"XXX>"
```

```
  If(kill) RETURN
....
  subjob= Nfld.EQ.1                      !request another interface?
....
  If(subjob) Then
    call SwGus_cleanup(line,m)           !no leading blanks, etc.
    lane= line
    m= INDEX(lane,' ')                   !first " "
    If(m.GT.1) lane(1:m)= ' '            !"YYYYY ..." => "..."
    If(SwGus_match(fld(1),'YyYy*y')) Then
      call pseudo_YyYyy(lane,OK)        !begin another interface "YYYYY>"
      Nopt=   0                         !already passed to another
    ElseIf(SwGus_match(fld(1),'Z*zz')) Then
      call pseudo_Zzz(lane,OK)          !begin another interface "ZZZ>"
....
    EndIf
  EndIf
....
  Do i=1,Nopt
    call SwGus_parse_I4_option(opt(i),op,Nval,val,string)
    if(SwGus_match(op,'HE*lp')) then
....
    elseif(SwGus_match(op,'STA*tus')) then
....
    else
      OK= .FALSE.
      call SwGus_squeeze(op,m)
      err= ':option "'//op(1:m)//'" not understood'
      call SwGus_add_path(OK,here,err)
      call SwGus_rep_err(OK,err)
    endif
  EndDo
....
ENDDO
RETURN
END
```

G Debugging and Profiling Routines

Typically, every routine contains the parameter **here** and a flag **bugON** to control debugging statements. The routine would then look something like:

```
SUBROUTINE SomeName(....)
....
CHARACTER*20 here
PARAMETER (here= 'SomeName')
logical bugON
....
CALL SwGus_enter(here, bugON)
IF(bugON) PRINT *,' debugging: '//here
...
CALL SwGus_exit(here)
RETURN
END
```

Subroutine SwGusTrace_BugHunt (where)
Character(*)* where

This routine adds one routine description to the list of files in which to turn on debugging statements. If *where* is blank, the list of descriptions is sent to IO channel 6, if "-", the list is erased, otherwise, *where* is added to the list. The list is case insensitive; the trailing wildcard "*" is supported. If *where* = "*", all debugging is enabled.

Subroutine SwGusTrace_dump (IO)
Character(*)* err
Logical OK

This routine writes to an already opened *IO* channel the current status, number of calls, and CPU time spent in all routines using **SwGusTrace_enter** and **SwGusTrace_exit** calls.

Subroutine SwGusTrace_enter (where, buggy)
Character(*)* where
Logical buggy

This routine compares the calling routine's name *where* to the list (from **SwGusTrace_BugHunt**) and sets *buggy*=.TRUE. if it matches an entry in the list. It always adds the entry and CPU time to the SwGusTrace internal table.

Subroutine SwGusTrace_exit (where)
Character(*)* where

This routine notes the exit from routine *where* to update the SwGusTrace internal table.

Subroutine SwGusTrace_reset ( )

This routine clears the SwGusTrace internal table.

Subroutine SwGusTrace_switch (ON_OFF)
Logical ON_OFF

This routine turns on and off all previously enabled (by **SwGusTrace_BugHunt**) debugging and all logging functions.

H CLAS-CODA 2.0 Related Routines

These routines use the **EvGen** routines, which in turn depend on **CODA** library calls. They will not be supported in the future.

Subroutine SwGus_add_sim_tag (first, last, Nmax, tag, OK, err)
Integer first, last, Nmax
Character(*)* tag(*), err
Logical OK

Fills the array *tag*, beginning with index *first*, with appropriate Ntuple tag names to contain the CLAS-CODA 2.0 event descriptor block. The number of particles is limited by *Nmax*, *last* is the index of the last *tag* used. The array *tag* is for use in creating an Ntuple via the **HBOOKN** call.

Subroutine SwGus_copy_event (idx1, buf1, ptr1, idx2, buf2, ptr2)
Integer idx1, buf1(*), ptr1(*), idx2, buf2(*), ptr2(*)

Copies an entire CLAS-CODA 2.0 event buffer *buf1* beginning at index *idx1* with pointer table *ptr1* into another buffer *buf2* which starts at index *idx2* and has pointer table *ptr2*.

Subroutine SwGus_fill_sim_tuple (bffr, first, last, Nmax, tuple, OK, err)
Integer bffr(*), first, last, Nmax
Real tuple(*)
Character(*)* err
Logical OK

Given a CLAS-CODA 2.0 buffer *bffr*, it extracts up to *Nmax* particle descriptions from the event descriptor block and puts them into the Ntuple array *tuple* beginning with index *first* before **HFN** is used. The *last* index used is returned.

Subroutine SwGus_got_event (ptr, buffer, ptrs, event_id, event_type, OK, err)
Integer ptr, buffer(*), ptrs(*), event_type
Character(*)* err
Logical OK

Returns one CLAS-CODA 2.0 event *buffer* beginning with *ptr* and described by by pointer table *ptrs*, ID#*event_id*, and type *event_type* from an already opened (by **EvGen_open**) file.