Writing Official H1SIM and H1REC Code
A How–To Approach

Benno List, ETH Zürich, IPP

29.1.2002, 1.7.2002

With the advent of the OO analysis framework and thanx to the fact that H1 simulation and reconstruction software is getting more and more complete, less and less people are faced with the necessity to write official H1SIM and H1REC code. For this reason, knowledge of the *dos* and *don'ts* of H1SIM and H1REC software is harder and harder to get.

This note is intended as a (sometimes random) collection of explanations, tips, tricks, dos and don'ts that anyone should know who starts to write H1 software that is intended to run within H1SIM or H1REC.

Topics covered are (hopefully one day) module steering, database access, short introductions to BOS, FPACK, LOOK, and an overview over what is to be found in H1UTIL.

Table of Contents

Chapter 1: The Canonical Main Program

The canonical code of a module

```
************************************************************************
      PROGRAM H1MAIN
************************************************************************

*     The BOS common /BCS/
*     This is normally created by using the sequence BCS,
*     but in the main program we want to set NBOSIW explicitly
      INTEGER      NHROW,NHCOL,NHLEN
      PARAMETER   (NHROW = 2, NHCOL = 1, NHLEN=2)
      INTEGER      NBOSIW
      PARAMETER   (NBOSIW=1000000)
      INTEGER      IW(NBOSIW)
      REAL         RW(NBOSIW)
      COMMON /BCS/ IW
      EQUIVALENCE (RW(1),IW(1))
      SAVE   /BCS/
+SELF, IF=IMPLNONE.
      INTEGER INTCHA,MBANK,MDARD,MDARS,MDROP,MLINK,MPRNT,MREADC,MSWAP
     +             ,NAMIND,NBANK,NDANR,NDROP,NLINK,NPRNT,NSWAP
+SELF.
      INTEGER   LW(NBOSIW)
      REAL      SW(NBOSIW)
      EQUIVALENCE (RW(1),IW(1),LW(1),SW(1))

*     The BOSMDL common /BOSMDL/
+SEQ,BOSMDL.

      PARAMETER (NBNAM=5000,NBPRT=1000)

*
*     Initialize BOS,   upto NBNAM bank names will be allowed.
*                       upto NBPRT banks can be printed with BPRNT
      CALL BNAMES(NBNAM)
      CALL BOS(IW,NBOSIW)
      IW(8) = NBPRT

*     read FPACK i/o commands from standard input (Unit 5)
      CALL FPARMR(5)
*     read BOS steering banks
      CALL BREADC

*     open access to database, initialize random number generator
      JOBNR=MDB('/JOB')
      CALL H1RNIN(JOBNR+1)

* ---- event processing loop -----------------------------------------
*
   10 CALL FSEQR ('BOSINPUT', IRET)
*       IRET >0 if event was read, =0 after last event, <0 thereafter
        IF(IRET.LT.0) GOTO 100

*
*       CALL ALL MODULES HERE!!!
```

```
*

*       Write out event to BOSOUTPUT, if BOSOUTPUT has been opened
        CALL FSEQW('BOSOUTPUT')

        GOTO 10
*
*     perform final book keeping

*     Close database
 100  IND=MDB('/END')

*     Close all FPACK files
      CALL FPARM('CLOSE ALL')
      END
```

The canonical H1 main program is basically the same for H1SIM, H1REC, and H1PHAN. The main difference between these applications is which modules are called in the event processing loop.

Examples of main programs are `//H1SIM/H1SIM/MAIN`, `//H1REC/H1REC/MAIN`, `//H1PHAN/MAIN/PHANH1`.

Chapter 2: Module steering

2.1 The Canonical Module

The canonical code of a module

```
************************************************************************
      SUBROUTINE MODNAME
************************************************************************
*
*HTMLP
*
*     Purpose:
*     --------
*
*HTMLI
*
*     Input:
*     ------
*
*HTMLO
*
*     Output:
*     -------
*
*HTMLE
*
*     Called by:
*     ----------
*
*     Calling:
*     --------
*
*     Author:
*     -------
*
*     Changed by:
*     -----------
*
************************************************************************

+SELF,IF=IMPLNONE.
      IMPLICIT NONE
+SELF.

C --- Arguments

C --- Variable declarations

+SEQ,BCS.
+SEQ,BOSMDL.
+SEQ,STFUDEF.

+SELF,IF=IMPLNONE.
      INTEGER     IVERSQ, IDATQQ
+SELF.
      CHARACTER*8 VERSQQ
```

```
C --- Data statements

C --- Statement function definitions

+SEQ,STFUNCT.

C --- Statements

+SEQ,VERSQQ.
+SEQ,DATEQQ.

*     Start module
      CALL MODULS ('MODNAME', IVERSQ, 'IBNK')

C --- ========================== BEGJOB ================================
      IF (BEGJOB) THEN
*       Do initialization at beginning of job
        IF (IDB .GT. 0) THEN
          PRINT *, 'MODNAME: BEGJOB, version ', IVERSQ,
     +             ', compiled at ', IDATQQ
        END IF

*       Define Bank formats of output banks
        CALL BKFMT ('OBNK', '2I,(I)')

      END IF

C --- ========================== ENDRUN ================================
      IF (ENDRUN) THEN
*       Finish whatever has to be finished for the last run (LCCRUN)
        IF (IDB .GT. 0) THEN
          PRINT *, 'MODNAME: ENDRUN ', LCCRUN
        END IF
      END IF

C --- ========================== BEGRUN ================================
      IF (BEGRUN) THEN
*       Do whatever has to be done at the beginning of a run,
        IF (IDB .GT. 0) THEN
          PRINT *, 'MODNAME: BEGRUN ', NCCRUN
        END IF

*       especially get data base banks

        CALL UGTBNK ('NAME', IND)

      END IF

C --- ========================== REVENT ================================
      IF (REVENT) THEN
*       This is a real event: Process it
        IF (IDB .GT. 0) THEN
          PRINT *, 'MODNAME: REVENT ', NCCRUN, NEVENT
        END IF

*       Don't forget to put output banks on the 'E' list!
```

```
        CALL BLIST (IW, 'E+', 'OBNK')

      END IF

C --- ========================= ENDJOB ===============================
      IF (ENDJOB) THEN
*       Do whatever has to be done at the beginning of a run,
        IF (IDB .GT. 0) THEN
          PRINT *, 'MODNAME: ENDJOB '
        END IF

      END IF

*     Finish module
      CALL MODULF

      RETURN
      END
```

2.2 Module initialization with MODULS

At the beginning of a module, the routine //H1UTIL/BOS_STUFF/MODULS must be called:

CALL **MODULS** (MODNAM, MODV, INNBNK)

```
CHARACTER*(*) MODNAM, INNBNK
INTEGER MODV
```

Purpose:
Steering and bookkeeping of modules:

- Count number of module calls.
- Determine obsolete input banks (those, which depend directly or indirectly on the output banks of the current module).
- Drop those banks, update banks, update list 'E' and update bank statistics of dropped banks.
- Add information of current module to MODS/MODI/MODO banks.
- Determine status of debug flag IDB of common /BOSMDL/.
- Set LOOK histogram area (area='module name',version=jobnr)
- Random number handling:

  Each module can request that the status of the random number generator is stored in the bank MODR at the start of a module for each event by calling routine MODULS with a negative version number. Then two different things can happen, depending on whether the steering card SAMERAND exists or not:

  – not existing: current status of random number generator filled into MODR bank, eventually over- writing previous state

– existing: the status stored in bank MODR for module MODNAM is used to define the status module MODNAM is used to define the status of the random number generator

Input:

MODNAM Name of the module

MODV Module version number (1.06/01 is coded as 10601)

INNBNK list of input bank names ('0' for empty list)

JOBNR Job number from common /CJOBNR/.

Output:

- Module name, version and index in commons /MODULN/, /MODULJ/.
- Module steering banks MODS, MODI, MODO
- Either updated MODR bank or updated random number status
- Updated MODC bank after reprocessing of modules
- Module statistics (Number of of calls, CPU time)
- Bank statistics (Number of of banks, size)

CALL **MODULT**

Purpose:
Print requested banks. Determine final length of MODS/MODI/MODO/MODR/MODC banks. Add =MDC bank (used banks from database) to MODC bank. Print timing and memory information Rewind random number status file. Check idle time of database connection.

Called from FSEQR at the end of each event.

CALL **MODULC**

Purpose:
Check bank length of all banks remaining in named bank area after a garbage collection.

CALL **MODUCX**

ENTRY of MODULC.

Purpose:
Print BOSCHECK result.

CALL **MODLEN**

Purpose:
Create or modify bank length for MODS,MODI,MODO,MODR banks.

`CALL` **MODLNX**

ENTRY of `MODLEN`.

Purpose:
Determines final bank length.

`CALL` **MODSTA** `(NAME, ITYPE, NAMM)`

```
CHARACTER*4 NAME
INTEGER ITYPE
CHARACTER*8 NAMM
```

Purpose:
Update bank statistics.

Input:

`NAME` name of bank

`ITYPE` ”type” of bank (new/old/dropped)

`NAMM` corresponding module name

`CALL` **PRTSTA**

ENTRY of `MODSTA`.

Purpose:
Print statistics.

`CALL` **MODMOD**

Purpose:
Handle module related statistics.

`CALL` **MODTIM**

ENTRY of `MODMOD`.

Purpose:
Called for bookkeeping of time and memory consumption at the end of a module.

`CALL` **MODTIX**

ENTRY of `MODMOD`.

Purpose:
Print timing information per event.

`CALL` **MODPRT**

ENTRY of `MODMOD`.

Purpose:
Print module statistics.

`CALL` **SETDEB** `(MODNAM, IDEB, ICOUNT)`

`CHARACTER*(*) MODNAM`
`IDEB, ICOUNT`
ENTRY of `MODMOD`

Purpose:
Define debug flags for module - for each module only the last call to SETDEB is relevant. The flags are typically set with a `DEBUG` steering card.

Input:

`MODNAM`	Module name
`IDEB`	Debug level (meaning depends on module, 0: no debugging)
`ICOUNT`	Number of events for which debug level is to be nonzero

Called by:
`FSEQST`.

`CALL` **MODINI**

ENTRY of `MODMOD`.

Purpose:
Prepare reinitialization of module steering.

`CALL` **JOBSUM**

Purpose:
Print job summary.

Called by: `FSEQR`

2.3 Dos and Don'ts for Modules

The caller of a module:

- **Must** call the module for *each* event.

  Not calling a module because an event is not flagged as REVENT, does not contain the necessary input banks, or for any other reason is a *very bad idea*.

  Such behaviour will cause problems when the module needs to perform certain actions when e.g. the BEGRUN or ENDRUN flags are set.

- **Must not** call one module from within another module (i.e. between the `MODULS` and `MODULF` calls in the module. This will completely screw up the bookkeeping of the module steering.

The Module itself:

- **Must** call `MODULS` and `MODULF` at the beginning and end of its code.

- **Must not** use any information from common blocks that are filled from routines that are not called by the module.

- **Must** use only information from BOS banks as input.
- **Must** write all relevant information necessary for further processing into BOS banks.
- **Must** call `BKFMT` for all banks that it produces.
- **Must** put all banks that it produces onto one of the BOS bank lists E (to be written out and dropped when the next event is loaded) or R (to be dropped without being written out) by calling `BLIST`.

Chapter 3: BOS: Banks, Tables, Steering Banks

3.1 The BOS Common

The BOS common /BCS/

```
*     The BOS common /BCS/
*     This is normally created by using the sequence BCS,
*     but in the main program we want to set NBOSIW explicitly
      INTEGER      NHROW,NHCOL,NHLEN
      PARAMETER   (NHROW = 2, NHCOL = 1, NHLEN=2)
      INTEGER      NBOSIW
      PARAMETER   (NBOSIW=1000000)
      INTEGER      IW(NBOSIW)
      REAL         RW(NBOSIW)
      COMMON /BCS/ IW
      EQUIVALENCE (RW(1),IW(1))
      SAVE   /BCS/
+SELF, IF=IMPLNONE.
      INTEGER INTCHA,MBANK,MDARD,MDARS,MDROP,MLINK,MPRNT,MREADC,MSWAP
     +             ,NAMIND,NBANK,NDANR,NDROP,NLINK,NPRNT,NSWAP
+SELF.
      INTEGER   LW(NBOSIW)
      REAL      SW(NBOSIW)
      EQUIVALENCE (RW(1),IW(1),LW(1),SW(1))
```

3.2 Initialization of BOS

CALL **BOS** (JW, NDIM)

INTEGER JW(*), NDIM

Purpose:
Initialization of the BOS system.

Input:

JW   Array for data storage.

NDIM Size of the array.

CALL **BNAMES** (NAMAX)

INTEGER NAMAX
Entry of BOS

Purpose:
Change default maximum number of names.

Mus be called *before* the call to BOS.

Input:

NAMAX Maximum number of names.

```
CALL BNRES
```

Entry of `BOS`

Purpose:
Increase the number of reserved words by 100.

Mus be called *before* the call to `BOS`.

3.3 Meaning of Special Words in the BOS Common

See `//BOS/BOS_MACROS/BLANKDEK`, sequence `CPARAM`.

`IW(4)` Print flag for `MREADC` ... (default: 1)

`IW(5)` Card input unit (default: 5)

`IW(6)` Print output unit (default: 6)

`IW(7)` Number of messages to be printed (default: 100)

`IW(8)` Number of banks to be printed (default: 100)

3.4 Reading in Steering Banks

3.5 Reading in Steering Banks

```
CALL BREADC
```

Purpose:
Read in steering cards from input unit given by `IW(5)`.

Reading is done until end of file is reached.

3.6 Access to Banks

```
IND = NLINK (NAME, NR)
```

```
CHARACTER*4 NAME
INTEGER NR
```

Purpose:
Get the index of a bank in the BOS common `/BCS/`. **Input:**

`NAME` Bank name.

`NR` Bank number.

Output:

`IND` Index to the bank.

To be added: NAMIND

3.7 Work Banks

3.8 Creating Tables with IADROW and IADFIN

```
IND = IADROW (NAME, NR, NCOL, ARR)
```

```
CHARACTER*4 NAME
INTEGER NR, NCOL
REAL ARR(*)
```

Purpose:
Function to add one row to a table. If this is the first row to add then the standard header of two words will be created first :

`IW(IND+1)`: `NCOL` Number of columns

`IW(IND+2)`: `NROW` Number of rows in table (=0 at beginning)

Then the row `ARR(1..NCOL)` is added to the table and the row counter `IW(IND+2)` is incremented. The table is identified by a name NAME and a number `NR`. The index `IADROW` to the workbank is returned. If `NCOL=0` on the fist call of `IADROW` for a given bank then the index 0 is returned (bank not yet existing) IF `NCOL` different from `NCOL` in table header the dat is ignored and a warning is written out.

Input:

`NAME` Bank name.

`NR` Bank number.

`NCOL` Number of columns to be added.

`ARR` Array with data words to be put into bank.

Output:

`IND` Index to temporary work bank in BOS workbank area. This index will be invaildated after each garbage collection in the work bank area, i.e. potentially after each call to `IADROW`.

```
IND = IADNRW (NAME, NR, NCOL, NROW, AR)
```

```
CHARACTER*4 NAME
INTEGER NR, NCOL, NROW
REAL ARR(*)
```

ENTRY of `IADROW`

Purpose:
Function to add several rows to a table. The actual argument for the array ARR can be either a 1 dimensional array containing the data row-by-row in consecutive order, or a two dimensional array defined in the calling routine as `ARR(NCOL,*)`.

Input:

`NAME` Bank name.

`NR` Bank number.

`NCOL` Number of columns to be added.

`NROW` Number of rows to be added.

`ARR` Array with data words to be put into bank.

Output:

`IND` Index to temporary work bank in BOS workbank area. This index will be invaildated after each garbage collection in the work bank area, i.e. potentially after each call to `IADROW` or `IADNRW`.

`IND = `**IADFIN**` (NAME, NR)`

```
CHARACTER*4 NAME
INTEGER NR
```

Purpose:
Final call to get named bank.
Input:

`NAME` Bank name.

`NR` Bank number.

Output:

`IND` Index to named bank in BOS common.

Chapter 4: FSEQR, FSEQW etc.

`CALL FSEQR (DANAME, IRET*)`

```
CHARACTER*(*) DANAME
INTEGER IRET
```

Input:

DANAME name of data set

Output:

IRET returned flag with values ...

- = 1 record read
- = 0 end-of-data condition - final loop thru modules
- =-1 after end-of data condition (skip modules)

Assuming: Standard array IW/RW in /BCS/.
BOS lists:

list E for input of RUNEVENT and RUNDATA records and output of RUNEVENT records (banks will *be dropped* after writing).

list S for output of RUNDATA records (banks will *not be dropped* after writing).

list R to delete banks before new input.

After end-of-data a RESTART of the reading loop is possible, either with another file (DANAME) or with the same file after a rewind (CALL FNRWND(DANAME)).
Remark:
The printing that is done by FSEQR is steered by the FPACK printing level, which is set by the FPACK steering card PRINT.
At the end, the following statistics are printed:

Printlevel at least	Routine
2	BOSTA: BOS statistics.
1	FPRFLS: FPACK file overview.
2	FPSTAT: FPACK input/output statistics.
0	JOBSUM: Job summary of module steering.
0	ERRLOG: Error message summary (only if IW(6).NE.0)

`CALL FSEQW (DANAME)`

```
CHARACTER*(*) DANAME
```

Purpose:
Mark present event for writing to output stream DANAME.
Input:

DANAME Output stream (normally 'BOSOUTPUT')

CALL **FSEQS**

Purpose:
User entry to skip present run.

CALL **FSEQE**

Purpose:
User entry to stop further reading.

CALL **FSEQST**

Purpose:
Initialization of FSEQR parameters.
Called by:
FSEQR, FSEQP, FSEQE, FSEQS at their first call.

CALL **FSEQF**

ENTRY of FSEQST
Purpose:
Forced restart.

4.1 Bank Lists

CALL **BLIST** JW, OPT, LIST

```
INTEGER JW(*)
CHARACTER*2 OPT
CHARACTER*(*) LIS
```

Input:

JW	BOS Array.
OPT	Option string, of form 'LO'
L	List name: one of CERST
O	Option: one of =+-
O='='	Set list
O='+'	Add to list
O='-'	Delete from list
LIST	List of bank names

Chapter 5: Error Logging

These routines are located in //H1UTIL/BOS_STUFF.

```
CALL ERRLOG  (ID, TEXT)
```

```
INTEGER ID
CHARACTER*(*) TEXT
```

Purpose:
Error logging facility.
Input:

ID	error identifier
TEXT	textstring connected to ID (ignored after first appearence of error ID)
NCCRUN, NEVENT	run and event number: from common /BOSMDL/
MODNA8, MODIND	module name and index: from common /MODULN/

Output: The following arrays are filled for each error message:

NAME	name of module producing the error message
IDERR	error identifier
IPSTRT	pointer to first character of message in ERRTXT
IPEND	pointer to last character
IRUN	run number of first event producing error
IEV	event number of first event producing error
ICNT	error count
ERRTXT	store of error messages (one big string)

```
CALL ERRSUM  (LUN)
```

```
INTEGER LUN
```
ENTRY of ERRLOG.
Purpose:
Print error summary on unit LUN.
Input:

LUN FORTRAN unit number for output.

Called by: FSEQR.

Chapter 6: Standard Steering Cards

6.1 DEBUG

With a steering card `DEBUG` a debug level can be set for a module for a number of events:

```
DEBUG  'MODULE' level events
```

Parameters:

`'MODULE'` Module name (5 to 8 characters).

`level` Debug level, which will be stored in the variable `IDB` in commonBCS (in `//H1UTIL/BOS_STUFF/BOSMAC`), when the corresponding module is active.

`events` Number of events during which the debugging is active. Afterwards, `IDB` will be set to 0.

If `IDB` is nonzero, error messages from `ERRLOG` will also be printed directly.

6.2 PRINT

```
PRINT  event1 -event2 'BANK' ['BNK2' ...]
```

Parameters:

`event1` Number of first event

`-event2` Negative number of last event

`'BANK'` Name of bank to be printed out (several banks can be given).

Chapter 7: DDL Descriptions

7.1 MODx Banks

7.1.1 MODS

Located in: //H1BANKS/DDL_GENERAL/MODS

DDL of MODS

```
!       BANKname BANKtype      !Comments
!
TABLE   MODS                        ! Module summary bank

!  attributes:

  1    MODNAM1     A                ! characters 1 to  4 of module name
  2    MODNAM2     A                ! characters 5 and 6 of module name
  3    MODVER      I   0  999999    ! module version (1.06/01 -> 10601)
  4    JOBNR       I                ! unique job number

END TABLE
```

7.1.2 MODR

Located in: //H1BANKS/DDL_GENERAL/MODR

DDL of MODR

```
!       BANKname BANKtype      !Comments
!
TABLE   MODR                          ! List of random number status

!  attributes:

   1    IRAND1      I                 ! 1. random number status word
   2    IRAND2      I
   3    IRAND3      I
   4    IRAND4      I
   5    IRAND5      I
   6    IRAND6      I
   7    IRAND7      I
   8    IRAND8      I
   9    IRAND9      I
  10    IRAND10     I
  11    IRAND11     I
  12    IRAND12     I
  13    IRAND13     I
  14    IRAND14     I
  15    IRAND15     I
  16    IRAND16     I
  17    IRAND17     I
  18    IRAND18     I
  19    IRAND19     I
```

```
  20    IRAND20      I
  21    IRAND21      I
  22    IRAND22      I
  23    IRAND23      I
  24    IRAND24      I
  25    IRAND25      I                ! 25. random number status word

 !  relations

  26     MODS      DMT1               ! relation to module

 END TABLE
```

7.1.3 MODO

Located in: //H1BANKS/DDL_GENERAL/MODO

DDL of MODO

```
 !       BANKname BANKtype      !Comments
 !
 TABLE   MODO                         ! List of output banks of modules

 !  attributes:

    1    OUTBANK      A               ! name of output bank

 !  relations

    2    MODS        DMT1             ! relation to module

 END TABLE
```

7.1.4 MODI

Located in: //H1BANKS/DDL_GENERAL/MODI

DDL of MODI

```
 !       BANKname BANKtype      !Comments
 !
 TABLE   MODI                         ! List of input banks of modules

 !  attributes:

    1    INBANK      A                ! name of input bank

 !  relations

   2    MODS        DMT1             ! relation to module

 END TABLE
```

7.1.5 MODC

Located in: //H1BANKS/DDL_GENERAL/MODC

DDL of MODC

```
!       BANKname BANKtype       !Comments
!
TABLE   MODC                         ! List of calibration banks used

!  attributes:

   1    CALBANK      A               ! name of calibration bank
   2    CALNTAC      I               ! NDB: -TAC (transaction ID)

!  relations

   3    MODS        DMT1             ! relation to module

END TABLE
```

Chapter 8: The Database

8.1 UGTBNK and UGTRUN

```
CALL UGTBNK (NAME, IND*)
```

```
CHARACTER*4 NAME
INTEGER IND
```

Purpose:
Interface to database access by IND=IABS(MDB(NAME)), plus: special version for C1J8 bank to allow for backward compatible treatment of this bank. To be used in H1REC (CJICAL) and event display (CJCXYZ). This should be a temporary fix. (for how long?)
Input:

NAME Database bank name.

Output:

IND Index to bank on BOS common.

```
CALL UGTRUN (NAME, IND*)
```

```
CHARACTER*4 NAME
INTEGER IND
```

Purpose:
Get run dependent banks for MC production, see MDBRUN for details.
Input:

NAME Database bank name.

Output:

IND Index to bank on BOS common.

```
IND = MDBRUN (NAME)
```

```
CHARACTER*4 NAME
```

Purpose:
Retrieve bank NAME from database for "DATA" section of the database. This call is used for all banks which are needed to simulate the real (degraded) H1 detector as function of the run number.
Input:

NAME Database bank name.

Output:

`IND` Index on BOS common.

`IND=0` means no bank found.

`IND<0` means same bank as in previous database access.

Called by:
`UGTRUN`

`CALL` **MODDEF** `(ICMZ)`

`INTEGER ICMZ`

Purpose:
Define default CMZ version to be used for database access for all further accesses to the database. Valid until overwritten by another default value or by `ICMZ=0`, which is equivalent to using the highest available version number. This is only done for MC data.

If `MODDEF` is e.g. called with `ICMZ=30500`, then any subsequent call of `UGTBNK` in a Monte Carlo event will return the highest version of the requested data base bank which has a CMZ version less or equal to 30500. All bank versions with higher CMZ version numbers will be ignored!

Remarks:

- `MODDEF` does nothing if no `HEAD` bank is present, or if the sixth word of the `HEAD` bank does not indicate that the event is a Montre Carlo event.
- Technically, `MODDEF` issues a call to the database NDB:
  `IND=MDB('DBPAR _GET NXE END VLE '//STR)`
  `STR` is initially set to `'99999'`, and is overwritten later with `ICMZ`, if `ICMZ .NE. 0`.

Input:

`ICMZ` CMZ version number to be used for database access

Chapter 9: Random Number Generation

9.1 Random Numbers between 0 and 1

`RND = H1RN ()`

Purpose:
Return one random number between 0 and 1 (excl.). This routine is based on the CERNLIB routine `RANLUX`.

Internally, `H1RN` just calls `H1RNV`.

Output:

`RND` Random number between 0 and 1

`CALL H1RNV (RVEC*, LENV)`

`REAL RVEC(*)`
`INTEGER LENV`

Purpose:
H1 random number generator which is an adaptation of cernlib package `RANLUX`Subtract-and-borrow random number generator proposed by Marsaglia and Zaman, implemented by F. James with the name RCARRY in 1991, and later improved by Martin Luescher in 1993 to produce "Luxury Pseudorandom Numbers". Fortran 77 coded by F. James, 1993).

In the H1 version, the default luxury level is set to the highest possible level of 4 .

For full details, see cernlib writeup, entry V115, taking into account the following renaming of entries:

H1 entry	ranlux entry
`H1RNV (...)`	`RANLUX(...)`
`H1RNIN (ISEED)`	`RLUXGO (LUX_DEFAULT, ISEED, 0, 0)`
`H1RNIV (...)`	`RLUXIN(...)`
`H1RNSV (...)`	`RLUXUT(...)`
`H1RNI4 (...)`	`RLUXGO(...)`
`H1RNS4 (...)`	`RLUXAT(...)`

Input:

`LENV` Number of requested random numbers

Output:

`RVEC` Vector of random numbers between 0 and 1

`CALL H1RNIN (ISEED)`

`REAL RVEC(*)`
`INTEGER LENV` ENTRY of `H1RNV`

Purpose:
Initialises the generator with one 32-bit integer setting the default luxury level.

Input:

`ISEED` Seed.

`CALL` **H1RNSV** `(ISVEC*)`

`INTEGER ISVEC(25)`
ENTRY of `H1RNV`

Purpose:
Outputs the current state of the generator with values of the 25 32-bit integer seeds, to be used for restarting with `H1RNIV`. `ISVEC` must be dimensioned 25 in the calling program .

Output:

`ISVEC` Vector of integers characterizing the generator state.

`CALL` **H1RNIV** `(ISVEC)`

`INTEGER ISVEC(25)`
ENTRY of `H1RNV`

Purpose:
Restarts the generator from vector `ISVEC` of 25 32-bit integers.

Input:

`ISVEC` Vector of integers characterizing the generator state.

`CALL` **H1RNS4** `LUX*, ISEED*, K1*, K2*`

`INTEGER LUX, ISEED, K1, K2`
ENTRY of `H1RNV`

Purpose:
Provides a more convenient but *less efficient* way to save the current state of the generator in 4 integers, by outputing the 4 numbers `LUX`, `ISEED`, `K1`, `K2`, which can be used later to restart the `H1RN` generator at the current point by calling `H1RNI4`. `K1` and `K2` specify how many numbers were generated since the initialization with `LUX` and `ISEED`. The restarting skips over `K1 + K2*10**9` numbers, so *it can be long*.

Output:

`LUX` Luxury level.

`ISEED` Original seed.

`K1` Number of calls modulo 10^9.

`K2` Number of calls divided by 10^9.

`CALL` **H1RNI4** `(LUX, ISEED, K1, K2)`

`INTEGER LUX, ISEED, K1, K2`
ENTRY of `H1RNV`

Purpose:
General form of `H1RN` initialisation with one 32-bit integer `ISEED` and setting Luxury Level `LUX` which is integer between zero and `MAXLEV`, or if `LUX .GT. 24`, it sets p=LUX directly. `K1` and `K2` should be set to zero unless restarting at a break point given by output of `H1RNS4`.

Input:

`LUX` Luxury level.

`ISEED` Original seed.

`K1` Number of calls modulo 10^9.

`K2` Number of calls divided by 10^9.

9.2 Gaussian Random Numbers

```
RND = H1RNG ()
```

Purpose:
Return one a random number distributed according to a Gaussian with width 1 and mean 0.

This routine internally calls `H1RNV` to get 2 uniform random numbers, and from that calculates two Gaussian random numbers. The first number is returned, the second stored and returned at the next call, unless `H1RNGI` is called first.

Output:

`RND` Gaussian random number.

```
CALL H1RNGI
```

ENTRY of `H1RNG`

Purpose:
Reset the internal cache of `H1RNG`.

```
CALL GAUSSV (G*, NG)
```

```
REAL G(*)
INTEGER NG
```

Purpose:
Return a vector of `NG` Gaussian Random numbers. `GAUSSV` internally calls `H1RNV` and implements the same algorithm as `H1RNG`.

Input:

`NG` Number of random numbers requested.

Output:

`G` Vector of Gaussian random numbers.

9.3 Setting the Seeds

Chapter 10: Various Tips and Tricks

Chapter 11: (Almost) Random Dos and Don’ts

11.0.1 No HBOOK

GEANT uses ZEBRA (CERN’s version of BOS) for its memory management; HBOOK does the same. For that reason, GEANT and HBOOK don’t mix easily, and in fact noone at H1 has yet succeeded to run GEANT and use HBOOK in the same program.

Since there is such a thing as H1SIMREC, a main program that runs GEANT, performs the DIGI step of the simulation and finally runs H1REC in one go (plus the trigger simulation, event classification and DST production), none of the modules that are intended to be run in this environment can use HBOOK. They have to use LOOK instead.

Sorry folks, that’s the way it is and why you have to learn a bit of LOOK in chapter 12.

Chapter 12: The first three minutes with LOOK for HBOOK Users

Entirely to be written.

Chapter 13: H1RAWD, POT, DST, and all that

Entirely to be written. Will probably have a “historic” HERA–1 part, and a newer HERA–2 part.

Bibliography

[1] Application Software Group: "CERN library,"
http://consult.cern.ch/shortwrups/,
http://wwwinfo.cern.ch/asdoc/cernlib.html.

[2] Stephan Egli: "BOS modules in H1 software — A set of rules and recommendations,"
H1 software note 12, Oct. 1990, updated march 1992.
http://pia.desy.de:8136/TWDM/cgi/twdmproto.pm/h1_publ/Get/ROOT/_2/_5/_4/19990208151531/sw012.txt.

[3] H.-U. Martyn et al: "Layout of banks and naming conventions for H1 software,"
H1 software note 13, Nov. 1990, updated May 1993.
http://pia.desy.de:8136/TWDM/cgi/twdmproto.pm/h1_publ/Get/ROOT/_2/_5/_4/19990208151822/sw013.txt.

Index