HNOTE 824

SELEX Minidst (VTUP/FTUP) Format for Pass2

J. Engelfried, B. Braunger, S.Y. Jun, I. Larin, V. Matveev

July 16, 2000

1 What are ftuples and vtuples?

Ftuples got invented long time ago, they are actually part of standard FORTRAN. We use them to overcome deficiencies of PAW ntuples (hence the name!), especially when the files are large. NTuple files are known to fail when 1) they are large (more than 128 MByte) and 2) there a several files to combine; tools like *nt/merge* and *nt/chain* do not work properly.

Within a program *producing* ntuples/ftuples, nearly nothing has to be changed. The *HBOOKN* statement has to be replaced with an *OPEN* statement to a unformatted Fortran file (see example below) and the *HFN* has to be replaced with an unformatted *WRITE* statement. That's all.

A program *using* ftuple files has to open the unformatted Fortran file, and has a loop with a unformatted read statement inside; additional statements like HFILL can be within the loop (see example later).

Vtuples are an extension of ftuples for pass2, with the same properties, except that we define entities of fixed length which make up a variable size event information.

2 Use of ftuple in SELEX for Pass1

At SELEX we use ftuples in combination with the recon-package within soap, but other applications are also possible, All the examples discussed here are coming from "recon"; to use it at other places, the examples have to be modified accordingly.

The use of ftuples (and ntuples) requires the knowledge of the structure of the array we use. PAW knows this by itself, but already using COMIS function requires the definitions. To facilitate this problem within SELEX, the file `recon_ntuple.inc` can be used within soap, in COMIS functions, and within ftuple routines. Like this, we have all the definitions used at one and only one location.

There is one more problem: We like to have as many ftuple files as possible on disk. A test showed that ftuple files with production output from pass1 are compressible to 50 % of their original size with gzip, which means that we can have double as many events on the same disk. The problem is that before reading in the file, it has to be unzipped. There is a special routine provided which will take care of this problem.

```
      subroutine recon_ntuple
#include "recon_ntuple.inc"                 !  recon ntuple definition
      Logical First
      Data First /.true./
x     If (First) then                       !  initalization
        First = .False.
        Open(10,file='recon.ftup1',Status='Unknown',Form='UNFORMATTED')
      EndIf
      tuple(01) = run_num
      tuple(02) = 100000*(event_num/10000000)
      ...here are 80 more lines ...
      Write(10) tuple
      Return
```

Table 1: Code Fragment extracted from `recon_ftuple.F` from pass1

2.1 How to produce an ftuple file

In table 2.1 we show some code fragments you need to produce an ftuple file. They are extracted from `recon_ntuple.F`.

2.2 How to use ftuple files in pass1

The use of these files is straight forward; if you know fortran, you can use them. There is a small example program, which you can actually find in `~jurgen/ftuple/anal1.F` on *fn781a* and *fsgi03*, and is also shown in table 2.2.

After the usual headers of a Fortran program, you can see the statement to include `recon_ntuple.inc`, and the initialization for HBOOK. You have to declare `ftup_open` as logical. The main part of the program is a double loop over several input files, and a loop over all the ftuple in this file. The routine `ftup_open` actually opens the file, and, if the file is gzipped, will unzip it "on the fly" via a "named pipe". You can deal with this as black box. The read statement fills the array (which is actually definded in `recon_ntuple.inc`) and you can do whatever you like; we fill here a histogram with the χ^2 of the primary vertex.

To compile and link this program, a `Makefile` is provided for your convinience. You can find it too in `~jurgen/ftuple/`. Just copy the examples to your directory, modify them as you like, and type `gmake`.

2.3 Other Utility programs for pass1

We wrote a few utility programs to ease working with ftuples. They are all in `$OFF781_BIN`. All programs will tell you how to use them by calling them without any arguments.

`ftup_select` will read ftuple files, apply some cuts, and write out ntuples and/or ftuples.

```
        Program anal1
        Implicit none
        Character*256 ftupfile
        Integer entries,PawSize,HMEMOR
#include "recon_ntuple.inc"
        Logical ftup_open
        Integer iargc,narg,blklen
        Integer i,j,k,n
        Parameter (PawSize=100000)
        COMMON/PAWC/HMEMOR(PawSize)
        If (iargc().lt.1) Then
        Write(*,*) 'Usage:  anal ftup_file(s)'
        Call Exit(1)
        EndIf
        Call HLIMIT(PawSize)
        Call HBOOK1(1,'primary vertex',100,0.,10.,0.)
        entries = 0
        Do narg = 1,iargc()
          Call getarg(narg,ftupfile)
          If (.not.ftup_open(10,ftupfile)) Goto 11
 10       Continue
            read(10,end=12,err=11) tuple
            If (run.le.0) Goto 12
            entries = entries + 1
            Call HFILL(1,pvx_chi2,0.,1.)
          Goto 10
 11       Continue
          Write(*,*) 'Error reading file ',ftupfile(:blklen(ftupfile)),
     &               ', continuing'
 12     Continue
        Close(10)
        EndDo
 999    Continue
        Close(10)
        Write(*,*) 'Read',entries,' Entries'
        Call HRPUT(0,'anal1.hbk','NT') ! write all histograms to file
        End
```

Table 2: Example program to analyze ftuple files from pass1

`ftup2ntup` converts one ftuple file to an ntuple.

`ftup_info` will read one ftuple file and either print a summary (similar to the paw nt/print command) or will print all or part of the array to the screen (similar to nt/scan).

2.4 The ftup format used in pass1

In pass1 we used a fixed length ftuple, designed primarily to hold information of tracks, vertices and recons. The definitions are shown in table 2.4.

3 Arguments for a new format

With the implementation of additional packages into soap, namely downstream vee/kink, vzero, kinks, and photon, all of them particpating at the recon level in reconstructing states, it was necessary to extend the format. Due the the variety of possible combination, a fixed format like in pass1 is not practical anymore. In addition, the old ftup format is restricted to a fixed number of 4 tracks, without any flexibility.

During the March 2000 SELEX meeting it was decided to implement a new, variable size format. A committe with the authors of this memo was formed, with the charge to design and implement the new format.

4 What info has to go into the vtup?

After a short discussion, we decided on the following info to go into the new structure, which is still reconstruction based: info about the recon, primary vertex, secondary vertex, tracks, ds kinks and vees, vzero, and photons. A detailed list of the separate entities is given in the next section. Most of the contents contained in ftup was conserved in the same order, just in separate blocks.

5 The vtup format used in pass2

The general idea is to have a "header" block with info about the recon itself, and afterwards blocks for the different entities. The last word in the recon information contains the (packed) information about how many blocks of every entity (also 0 is possible!) are following. A summary is shown in table 4.

The structure of all the blocks is given in tables 5 to 11

5.1 Contents of the V0 block

The information from V0 is more complicated and cannot be presented in one simple table. A description follows here:

01	run	run number
02	event	100000*processor+event
03	bitsor	event reconstr, ored bits
		primary vertex [I*4]
04	pvx_x	x_vertex [cm]
05	pvx_y	y_vertex [cm]
06	pvx_z	z_vertex [cm]
07	pvx_sx	x_vertex error [cm]
08	pvx_sy	y_vertex error [cm]
09	pvx_sz	z_vertex error [cm]
10	pvx_chi2	reduced χ^2
11	pvx_nf	tracks in fit
12	pvx_nt	tracks in event
		secondary vertex
13	svx_x	x_vertex [cm]
14	svx_y	y_vertex [cm]
15	svx_z	z_vertex [cm]
16	svx_sx	x_vertex error [cm]
17	svx_sy	y_vertex error [cm]
18	svx_sz	z_vertex error [cm]
19	svx_chi2	reduced χ^2
20	svx_isol	vertex isolation distance
		beam track
21	btk_x0	x0 in vx_spec [cm]
22	btk_y0	y0 in vx_spec [cm]
23	btk_px	px in vx_spec [GeV/c]
24	btk_py	py in vx_spec [GeV/c]
25	btk_pz	pz0 in vx_spec [GeV/c]
26	btk_pid	pid word
27	btk_chi2	segment reduced χ^2
28	btk_type	q*[1000*nplanes+track_type]
		1st secondary vertex track
29	tk1_x0	x0 in vx_spec [cm]
30	tk1_y0	y0 in vx_spec [cm]
31	tk1_px	px in vx_spec [GeV/c]
32	tk1_py	py in vx_spec [GeV/c]
33	tk1_pz	pz in vx_spec [GeV/c]
34	tk1_pid	pid word
35	tk1_chi2	trk reduced χ^2
36	tk1_type	q*[1000*nplanes+track_type]
		2nd secondary vertex track
37	tk2_x0	x0 in vx_spec [cm]
38	tk2_y0	y0 in vx_spec [cm]
39	tk2_px	px in vx_spec [GeV/c]
40	tk2_py	py in vx_spec [GeV/c]
41	tk2_pz	pz in vx_spec [GeV/c]
42	tk2_pid	pid word
43	tk2_chi2	trk reduced χ^2
44	tk2_type	q*[1000*nplanes+track_type]
		3rd secondary vertex track
45	tk3_x0	x0 in vx_spec [cm]
46	tk3_y0	y0 in vx_spec [cm]
47	tk3_px	px in vx_spec [GeV/c]
48	tk3_py	py in vx_spec [GeV/c]
49	tk3_pz	pz in vx_spec [GeV/c]
50	tk3_pid	pid word
51	tk3_chi2	trk reduced χ^2
52	tk3_type	q*[1000*nplanes + track_type]
		4th secondary vertex track
53	tk4_x0	x0 in vx_spec [cm]
54	tk4_y0	y0 in vx_spec [cm]
55	tk4_px	px in vx_spec [GeV/c]
56	tk4_py	py in vx_spec [GeV/c]
57	tk4_pz	pz in vx_spec [GeV/c]
58	tk4_pid	pid word
59	tk4_chi2	trk reduced χ^2
60	tk4_type	q*[1000*nplanes + track_type]
61	id	id of recon definition
62	status	100*# recons + # this recon
63	tgt	target # ass. to prim. vertex
64	l	z distance from prim. vertex
65	sigma	l uncertainty
66	q	charge of recon
67	e	energy
68	px	px momentum
69	py	py momentum
70	pz	pz momentum
71	mass	effective mass
72	dmass	effective mass uncertainty
73	pvtx	point-back reduced χ^2
74	chi2	tgt-prim_vtx reduced χ^2
75	ndof	tgt-prim_vtx dof
76	bit	mode bits
77	pmtrk1	pid, mass ass., trk index [I*4]
78	pmtrk2	pid, mass ass., trk index [I*4]
79	pmtrk3	pid, mass ass., trk index [I*4]
80	pmtrk4	pid, mass ass., trk index [I*4]
81	pmtrk5	pid, mass ass., trk index [I*4]
82	pmtrk6	pid, mass ass., trk index [I*4]

pid word:
each decimal digit is 10*ratio

	e	μ	π	K	p	Σ
975310	.9	.7	.5	.3	.1	.0

Table 3: Fixed length ftuple used in pass1

Entity	Length
Recon	20
prim. vertex	9
sec. vertex	8
beam track	8
vertex tracks	9
vzero	20
downstream	15
photons	9

Table 4: Table of entities and their lengths (in 32bit words) used in vtup

	Name	Type	Description
1	run	Int	run number
2	event	Int	100000*processor+event
3	bitsor	Int	event reconstr, ored bits
4	id	Int	id of recon definition
5	status	Int	100*# recons + # this recon
6	tgt	Int	target # ass. to prim. vertex
7	l	Real	z distance from prim. vertex
8	sigma	Real	l uncertainty
9	q	Real	charge of recon
10	e	Real	energy
11	px	Real	px momentum
12	py	Real	py momentum
13	pz	Real	pz momentum
14	mass	Real	effective mass
15	dmass	Real	effective mass uncertainty
16	pvtx	Real	point-back reduced χ^2
17	chi2	Real	tgt-prim_vtx reduced χ^2
18	ndof	Real	tgt-prim_vtx dof
19	bit	Int	mode bits
20	nentity	Int	packed number of γdvtbsp
			(p=# prim_vert, s=# sec_vert, etc)

Table 5: Contents of the "header" or "recon" block

	Name	Description	
1	pvx_x	Real	x_vertex [cm]
2	pvx_y	Real	y_vertex [cm]
3	pvx_z	Real	z_vertex [cm]
4	pvx_sx	Real	x_vertex error [cm]
5	pvx_sy	Real	y_vertex error [cm]
6	pvx_sz	Real	z_vertex error [cm]
7	pvx_chi2	Real	reduced χ^2
8	pvx_nf	Int	tracks in fit
9	pvx_nt	Int	tracks in event

Table 6: Contents of the primary vertex block

	Name	Description
1	svx_x	x_vertex [cm]
2	svx_y	y_vertex [cm]
3	svx_z	z_vertex [cm]
4	svx_sx	x_vertex error [cm]
5	svx_sy	y_vertex error [cm]
6	svx_sz	z_vertex error [cm]
7	svx_chi2	reduced χ^2
8	svx_isol	vertex isolation distance

Table 7: Contents of the secondary vertex block (all Real)

	Name	Type	Description
1	btk_x0	Real	x0 in vx_spec [cm]
2	btk_y0	Real	y0 in vx_spec [cm]
3	btk_px	Real	px in vx_spec [GeV/c]
4	btk_py	Real	py in vx_spec [GeV/c]
5	btk_pz	Real	pz0 in vx_spec [GeV/c]
6	btk_pid	Int	pid word
7	btk_chi2	Real	segment reduced χ^2
8	btk_type	Int	q*[1000*nplanes+track_type]

Table 8: Contents of the block containing beam track information

	Name	Type	Description
1	tk_x0	Real	x0 in vx_spec [cm]
2	tk_y0	Real	y0 in vx_spec [cm]
3	tk_px	Real	px in vx_spec [GeV/c]
4	tk_py	Real	py in vx_spec [GeV/c]
5	tk_pz	Real	pz in vx_spec [GeV/c]
6	tk_pid	Int	pid word (RICH e,μ,π,K,p, Σ,Ξ,Ω,ETRDe)
7	tk_chi2	Real	trk reduced χ^2
8	tk_type	Int	q*[1000*nplanes+track_type]
9	pmtrk	Int	pid, mass ass., trk index

Table 9: Contents of the block containing vertex track information

	Name	Type	Description
1	ds_status	Int	7 Both trks have p (best uncertainty in p)
			6/5 one trk has reconstructed p
			4 both trks have reconstructed p (worst uncertainty)
			bits above 8: See vee_stub code for explanation
2	ds_x01	Real	x_0 of branch at decay point
3	ds_y01	Real	y_0 of branch at decay point
4	ds_px1	Real	p_x of branch at decay point
5	ds_py1	Real	p_y of branch at decay point
6	ds_pz1	Real	p_z of branch at decay point
7	ds_pid1	Int	RICH pid (see pid word in table 9)
8	ds_curv1	Real	q*curv of this branch (may be reconstructed)
9	ds_x02	Real	x_0 of branch at decay point
10	ds_y02	Real	y_0 of branch at decay point
11	ds_px2	Real	p_x of branch at decay point
12	ds_py2	Real	p_y of branch at decay point
13	ds_pz2	Real	p_z of branch at decay point
14	ds_pid2	Int	RICH pid (see pid word in table 9)
15	ds_curv2	Real	q*curv of this branch (may be reconstructed)

Table 10: Contents of the block containing info from the ds package. In the case if a vee, words 2-8 correspond to the positive and 9-15 to the negative daughter track. In the case if a kink, words 2-8 correspond to the parent and 9-15 to the daughter track.

	Name	recon with single γ	recon with π^0
1	g_x0	x_adcgam	1024 * nint((x_adcgam1+128.)*4.) + nint((x_adcgam2+128.)*4.)
2	g_y0	y_adcgam	1024 * nint((y_adcgam1+128.)*4.) + nint((y_adcgam2+128.)*4.)
3	g_z0	Spare	Spare
4	g_px	p_x of γ	p_x of π^0
5	g_py	p_y of γ	p_y of π^0
6	g_pz	p_z of γ	p_z of π^0
7	g_pid	dime_adcgam, status_adcgam	energy_adcgam1/max(0.001,energy_adcgam2)
8	g_chi2	chi2_adcgam, nadcgam	dmass of π^0, π^0 multiplicity
9	g_type	npht_adcgam, type_adcgam	npht_adcgam1, type_adcgam1, npht_adcgam2, type_adcgam2

Table 11: Contents of the block containing photon information (all Real)

ftup_sv creates list of integer parameters io_buf(1:length), with length ≤ 19. This structure gets copied directly into the V0-block within the vtuple, with the length prepended, leading to a total length of 20 word for this block.

The last word keeps 2 parameters of sv_recn itself:

dbs_svr: recn vector miss distance to beam track;
dbs_svr = REAL(MOD(io_buf(length),10000))/10000.

dbz_svr: z recn - z closest approach to beam line;
dbz_svr = REAL(io_buf(length)/10000)/200.

Other words keep the parameters of last track in pid_recdf notation. It can be strange particle or anti-particle of following types:

vee: Λ or K_s (from:v0) (table 12)
sig0: $\Lambda + \gamma$ recn (from:gr) (table 13)
recn: any recn except recn produced by recon_v0 & recon_gr (table 14)
kink: Σ^{+-}, Ξ^-, Ω^- (table 15)
sir: Σ in RICH (table 16)
cslt: Ξ^- or Ω^- with Λ, meson in m1 and hyperon in vertex (table 17)
csl: Ξ^- or Ω^- with Λ and meson in m1 (table 18)
ctl: Ξ^- or Ω^- with Λ and hyperon track in vertex (table 19)

The contents of the block depends on the different types and is described in tables 12 to 19

6 How to convert vtup into ntuple

A program called *vtup2ntup* is provided to convert vtuple files into column wise ntuples. Please see `$OFF871_INC/recon_vtuple.inc` for information about the structure. There is a

1	length	8
2	status_v0	pattern: IAND(status_v0,vee_s_v0).ne.0 see definition in v0_cbk & v0_bk; in addition, bit "fail_s_v0" is set if any of decay products is electron in photon detector
3	ipid	rich info about highest momentum track rich id = ipid/100000000 -1 (-1:no rich, 0:bgr, else rich hypo) prob [%] i = MOD(ipid,100) k = MOD(ipid/100,100) p = MOD(ipid/10000,100) s = MOD(ipid/1000000,100)
4	trk_index	itrk = MOD(trk_index,100) (first, highest momentum) ktrk = MOD(trk_index/100,100) (second may have no momentum) vee mass shift from PDT value [GeV/cc] dm = REAL(trk_index/10000000)/1000.-0.050 p(itrk)/p(vee) = REAL(MOD(trk_index/10000,100))/1000.
5	word	p(vee) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data) Netr = word/100000000-1
6	word	dca(vee) = REAL(MOD(word,10000))/10000. χ^2(dca(vee)) = REAL(MOD(word/10000,100)) θ(vee) = REAL(word/1000000)/1000.
7	word	vee L/σ from primary vertex dL = REAL(word/1000000)/10. vee miss angle to primary vertex unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
8	miss	miss distance of nearest track to vee vertex dnear = REAL(MOD(miss,10000))/10000. χ^2(dnear) = REAL(miss/10000)
9	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 12: Contents of the block containing information for a vee from V0

1	length	11
2	status_v0	pattern: IAND(status_v0,99).eq.12) .and. IAND(status_v0, vee_s_v0).eq.0 .and. IAND(status_v0,kink_s_v0).eq.0 .and. IAND(status_v0, csl_s_v0).eq.0 .and. IAND(status_v0, ctl_s_v0).eq.0 photon energy Energy_adcgam = REAL(MOD(status_v0/100,10000))/10.
3	adcgam	photon Detector: npht_adcgam = MOD(adcgam,10) shower χ^2: Chi2_adcgam = REAL(status_v0/10000)/10.
4	word	Σ^0 mass = REAL(word,10000)/1000. Σ^0 momentum = REAL(word/10000)/10.
5	status_v0	pattern: IAND(status_v0,vee_s_v0).ne.0 see definition in v0_cbk & v0_bk; in addition, bit "fail_s_v0" is set if any of decay products is electron in photon detector
6	ipid	rich info about highest momentum track rich id = ipid/100000000 -1 (-1:no rich, 0:bgr, else rich hypo) prob [%]: i = MOD(ipid,100), k = MOD(ipid/100,100) p = MOD(ipid/10000,100), s = MOD(ipid/1000000,100)
7	trk_index	itrk = MOD(trk_index,100) (first, highest momentum) ktrk = MOD(trk_index/100,100) (second may have no momentum) vee mass shift from PDT value [GeV/cc] dm = REAL(trk_index/10000000)/1000.-0.050 p(itrk)/p(vee) = REAL(MOD(trk_index/10000,100))/1000.
8	word	p(vee) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data) Netr = word/100000000-1
9	word	dca(vee) = REAL(MOD(word,10000))/10000. χ^2(dca(vee)) = REAL(MOD(word/10000,100)) θ(vee) = REAL(word/1000000)/1000.
10	word	vee L/σ from primary vertex dL = REAL(word/1000000)/10. vee miss angle to primary vertex unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
11	miss	miss distance of nearest track to vee vertex dnear = REAL(MOD(miss,10000))/10000. χ^2(dnear) = REAL(miss/10000)
12	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 13: Contents of the block containing information for a Σ^0 from V0

1	length	2
2	status_v0	pattern: IAND(status_v0,99).eq.15) .and. IAND(status_v0, vee_s_v0).eq.0 .and. IAND(status_v0,kink_s_v0).eq.0 .and. IAND(status_v0, csl_s_v0).eq.0 .and. IAND(status_v0, ctl_s_v0).eq.0 recn mass = REAL(status_v0/100)/1000.
3	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 14: Contents of the block containing information for any recn except vee and Σ^0 from V0

file size limit on ntuple files (about 100 MBytes), which might prevent you converting a large vtup file directly. It is possible to select a subset of the vtuple to reduce the output size.

6.1 Converting from vtup to ftup format

A routine with name *vtup2ftup* is provided which will convert the new variable size vtup format to the ftup format used during pass1. Information not available in ftup format is obviously lost during the process. This file can be converted via *ftup2ntup* into fixed length ntuple.

1	length	7 (13,19) for 1 (2,3) hypothesis
2	status_v0	pattern: IAND(status_v0,kink_s_v0).ne.0); see further definition in v0_cbk & v0_rbk
3	ipid	rich info about track in m2 spec; see ipid definition in table 12
4	word	momentum ratio pd/ph: p(itrk)/p(ktrk) = REAL(MOD(word/10000,1000))/1000. angle in [YZ] plane: dty = REAL(MOD(word/10000000,100))/10000.
5	word	hyperon momentum: p(ktrk) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data): Netr = word/100000000-1
6	word	dca(kink vertex) = REAL(MOD(word,10000))/10000. χ^2(dca) = REAL(MOD(word/10000,100)) θ(kink vertex) = REAL(word/1000000)/1000.
7	single	dca(single track hypo at Z(kink vertex)) = REAL(MOD(single,10000))/10000. χ^2(dca(single track hypo)) = REAL(MOD(single/10000,100)) θ(single track hypo) = REAL(single/1000000)/1000.
8	status_v0	pattern for 1 alternative hypo (particles only): length=13 and IAND(status_v0,anti_s_v0).eq.0) pattern for 1 alternative hypo (anti-particles): length=13 and IAND(status_v0,anti_s_v0).ne.0)
9	ipid	rich info about track in m2 spec; see ipid definition in table 12
10	word	momentum ratio pd/ph: p(itrk)/p(ktrk) = REAL(MOD(word/10000,1000))/1000. angle in [YZ] plane: dty = REAL(MOD(word/10000000,100))/10000.
11	word	hyperon momentum: p(ktrk) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data) Netr = word/100000000-1
12	word	dca(kink vertex) = REAL(MOD(word,10000))/10000. χ^2(dca) = REAL(MOD(word/10000,100)) θ(kink vertex) = REAL(word/1000000)/1000.
13	single	dca(single track hypo at Z(kink vertex)) = REAL(MOD(single,10000))/10000. χ^2(dca(single track hypo)) = REAL(MOD(single/10000,100)) θ(single track hypo) = REAL(single/1000000)/1000.
14	status_v0	pattern for 2 alternative hypos (particles only): length=19 and IAND(status_v0,anti_s_v0).eq.0) pattern for 2 alternative hypos (anti-particles): length=19 and IAND(status_v0,anti_s_v0).ne.0)
15	ipid	rich info about track in m2 spec; see ipid definition in table 12
16	word	momentum ratio pd/ph: p(itrk)/p(ktrk) = REAL(MOD(word/10000,1000))/1000. angle in [YZ] plane: dty = REAL(MOD(word/10000000,100))/10000.
17	word	hyperon momentum: p(ktrk) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data): Netr = word/100000000-1
18	word	dca(kink vertex) = REAL(MOD(word,10000))/10000. χ^2(dca) = REAL(MOD(word/10000,100)) θ(kink vertex) = REAL(word/1000000)/1000.
19	single	dca(single track hypo at Z(kink vertex)) = REAL(MOD(single,10000))/10000. χ^2(dca(single track hypo)) = REAL(MOD(single/10000,100)) θ(single track hypo) = REAL(single/1000000)/1000.
20	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 15: Contents of the block containing information about kinks from V0

1	length	3
2	status_v0	pattern: IAND(status_v0,99).eq.13) .and. IAND(status_v0, vee_s_v0).eq.0 .and. IAND(status_v0,kink_s_v0).eq.0 .and. IAND(status_v0, csl_s_v0).eq.0 .and. IAND(status_v0, ctl_s_v0).eq.0 hyperon momentum p = REAL(MOD(status_v0/100,10000))/10. Number of etrd hits for hyperon track (-1:no data) Netr = status_v0/1000000-1
3	ipid	rich info about track in m2 spec (see ipid definition in table 12)
4	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 16: Contents of the block containing information about a Σ in the RICH from V0

1	length	15
2	status_v0	pattern: IAND(status_v0,csl_s_v0).ne.0 .and. MOD(word3,100).ne.0 (Ξ^- or Ω^- hyperon track index) see definition in v0_cbk & v0_bk
3	mpid	rich info about meson track
4	word3	itrk = MOD(word3,100) (Ξ^- or Ω^- track index) ktrk = MOD(word3/100,100) (meson track index) effective mass shift from PDT value [GeV/cc] = REAL(word3/10000000)/1000.-0.050 p(Λ)/p(hyperon) = REAL(MOD(word3/10000,100))/1000.
5	word	p(hyperon) = REAL(MOD(word,10000))/10. Z(Λ+meson vertex) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for meson track (-1:no data): Netr = word/100000000-1
6	word	dca(Λ+meson vertex) = REAL(MOD(word,10000))/10000. χ^2(dca) = REAL(MOD(word/10000,100)) θ(Λ,meson) = REAL(word/1000000)/1000.
7	word	Hyperon L/σ from primary vertex: dL = REAL(word/1000000)/10. Hyperon miss angle to primary vertex: unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
8	word	miss distance of nearest track to Hyperon decay vertex dnear = REAL(MOD(word,10000))/10000. χ^2(dnear) = REAL(word/10000)
9	status_v0	same as word 2 in table 12
10	ipid	same as word 3 in table 12
11	trk_index	itrk = MOD(trk_index,100) (first, highest momentum) ktrk = MOD(trk_index/100,100) (second may have no momentum) vee mass shift from PDT value [GeV/cc]: dm = REAL(trk_index/10000000)/1000.-0.050 p(itrk)/p(vee) = REAL(MOD(trk_index/10000,100))/1000.
12	word	p(vee) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data): Netr = word/100000000-1
13	word	dca(vee) = REAL(MOD(word,10000))/10000. χ^2(dca(vee)) = REAL(MOD(word/10000,100)) θ(vee) = REAL(word/1000000)/1000.
14	word	vee L/σ from primary vertex: dL = REAL(word/1000000)/10. vee miss angle to primary vertex: unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
15	miss	miss distance of nearest track to vee vertex: dnear = REAL(MOD(miss,10000))/10000. χ^2(dnear) = REAL(miss/10000)
16	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 17: Contents of the block containing information about a Ξ^- or Ω^- with Λ, meson in m1 and hyperon in vertex from V0

1	length	15
2	status_v0	pattern: IAND(status_v0,csl_s_v0).ne.0 .and. MOD(word3,100).eq.0 (Ξ^- or Ω^- track missing) see definition in v0_cbk & v0_bk
3	mpid	rich info about meson track
4	word3	itrk = MOD(word3,100) (Ξ^- or Ω^- track index) ktrk = MOD(word3/100,100) (meson track index) effective mass shift from PDT value [GeV/cc] = REAL(word3/10000000)/1000.-0.050 p(Λ)/p(hyperon) = REAL(MOD(word3/10000,100))/1000.
5	word	p(hyperon) = REAL(MOD(word,10000))/10. Z(Λ+meson vertex) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for meson track (-1:no data): Netr = word/100000000-1
6	word	dca(Λ+meson vertex) = REAL(MOD(word,10000))/10000. χ^2(dca) = REAL(MOD(word/10000,100)) θ(Lambda,meson) = REAL(word/1000000)/1000.
7	word	Hyperon L/σ from primary vertex: dL = REAL(word/1000000)/10. Hyperon miss angle to primary vertex: unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
8	word	miss distance of nearest track to Hyperon decay vertex dnear = REAL(MOD(word,10000))/10000. χ^2(dnear) = REAL(word/10000)
9	status_v0	same as word 2 in table 12
10	ipid	same as word 3 in table 12
11	trk_index	itrk = MOD(trk_index,100) (first, highest momentum) ktrk = MOD(trk_index/100,100) (second may have no momentum) vee mass shift from PDT value [GeV/cc]: dm = REAL(trk_index/10000000)/1000.-0.050 p(itrk)/p(vee) = REAL(MOD(trk_index/10000,100))/1000.
12	word	p(vee) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data): Netr = word/100000000)-1
13	word	dca(vee) = REAL(MOD(word,10000))/10000. χ^2(dca(vee)) = REAL(MOD(word/10000,100)) θ(vee) = REAL(word/1000000)/1000.
14	word	vee L/σ from primary vertex: dL = REAL(word/1000000)/10. vee miss angle to primary vertex: unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
15	miss	miss distance of nearest track to vee vertex: dnear = REAL(MOD(miss,10000))/10000. χ^2(dnear) = REAL(miss/10000)
16	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 18: Contents of the block containing information about a Ξ^- or Ω^- with Λ, meson in m1 from V0

1	length	15
2	status_v0	pattern: IAND(status_v0,csl_s_v0).ne.0
3	mpid	not used
4	word3	itrk = MOD(word3,100) (Ξ^- or Ω^- track index) ktrk = MOD(word3/100,100) (meson track index) effective mass shift from PDT value [GeV/cc] = REAL(word3/10000000)/1000.-0.050 p(Λ)/p(hyperon) = REAL(MOD(word3/10000,100))/1000.
5	word	p(hyperon) = REAL(MOD(word,10000))/10. Z(Λ+meson vertex) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for meson track (-1:no data): Netr = word/100000000-1
6	word	dca(Λ+meson vertex) = REAL(MOD(word,10000))/10000. χ^2(dca) = REAL(MOD(word/10000,100)) θ(Λ,meson) = REAL(word/1000000)/1000.
7	word	Hyperon L/σ from primary vertex: dL = REAL(word/1000000)/10. Hyperon miss angle to primary vertex: unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
8	word	miss distance of nearest track to Hyperon decay vertex: dnear = REAL(MOD(word,10000))/10000. χ^2(dnear) = REAL(word/10000)
9	status_v0	same as word 2 in table 12
10	ipid	same as word 3 in table 12
11	trk_index	itrk = MOD(trk_index,100) (first, highest momentum) ktrk = MOD(trk_index/100,100) (second may have no momentum) vee mass shift from PDT value [GeV/cc]: dm = REAL(trk_index/10000000)/1000.-0.050 p(itrk)/p(vee) = REAL(MOD(trk_index/10000,100))/1000.
12	word	p(vee) = REAL(MOD(word,10000))/10. Z(vee) - z_recn = REAL(MOD(word/10000,10000))/10. Number of etrd hits for first track (-1:no data): Netr = word/100000000-1
13	word	dca(vee) = REAL(MOD(word,10000))/10000. χ^2(dca(vee)) = REAL(MOD(word/10000,100)) θ(vee) = REAL(word/1000000)/1000.
14	word	vee L/σ from primary vertex: dL = REAL(word/1000000)/10. vee miss angle to primary vertex: unco = REAL(MOD(word,10000))/10000. χ^2(unco) = REAL(MOD(word/10000,100))
15	miss	miss distance of nearest track to vee vertex: dnear = REAL(MOD(miss,10000))/10000. χ^2(dnear) = REAL(miss/10000)
16	last	recn vector miss distance to beam track: dbs_svr = REAL(MOD(last,10000))/10000. Z recn - Z closest approach to beam line: dbz_svr = REAL(last/10000)/200.

Table 19: Contents of the block containing information about a Ξ^- or Ω^- with Λ and hyperon track (meson not found) from V0