DELPHI Collaboration DELPHI 97-61 PROG 220
24 June, 1997

Kinematic Analysis Language - A User Manual

A.W. Borgland, G. Eigen
Department of Physics, University of Bergen

Abstract

KAL - *Kinematic Analysis Language* - is a modern tool for data analysis in experimental particle physics which has become available now for use in DELPHI. This DELPHI note is a User Manual and is meant to be a first introduction to KAL. It will give the motivation for introducing such a tool, and outline the basic ideas behind it. All the features will be explained in detail, and a complete example of an analysis written now within the KAL framework will be given. It will also explain practical details how to get KAL running both at CERN and locally at your institute.

Contents

1 Introduction

KAL - *Kinematic Analysis Language* - is a modern tool for data analysis in experimental particle physics which has become available now for use in DELPHI. This DELPHI note is a User Manual and is meant to be a first introduction to KAL. It will give the motivation for introducing such a tool, and outline the basic ideas behind it. All the features will be explained in detail, and a complete example of an analysis written within the KAL framework will be given. It will also explain practical details how to get KAL running locally. A complete and updated set of WWW-pages exists and can be accessed at the following address:

http://wwwcn.cern.ch/~borgland/kal.html

In particular, these WWW-pages will have the function of a reference manual and contain the complete syntax of all the KAL features, in addition to a complete list of variables available in KAL. Even though most of the information can be found in the appendices of this DELPHI note, the user is recommended to actively use these WWW-pages to get more details and the latest changes.

2 Motivation

Physics analyses in multipurpose detectors such as DELPHI have become rather complex, since event multiplicities at the Z^0 are rather large, a lot of quantities are measured for each particle, and rather sophisticated selection criteria are necessary to separate the signal from backgrounds. Typically, a lot of data have to be processed, manipulated and managed before the desired results are obtained. Combinatorics issues arise as usually several possibilities exist for reconstructing a particular final state. To write an analysis program from scratch in FORTRAN is a rather time consuming effort. It also requires good book keeping to avoid double counting and minimize the possibility for mistakes. It is also frequently tedious to extract the information needed from the global data structures. Additional time is required to debug the code, since mistakes can be easily made. Thus an analysis tool which handled book keeping and provided all information in an easily accessible place would be rather suitful. Such a tool actually exists. It is called KAL (*Kinematic Analysis Language*) and was first developed for ARGUS by Hartwig Albrecht [1]. KAL takes care of booking. It handles combinatorics of up to four-particle final states, it determines intermediate resonances, it finds primary, secondary and tertiary vertices, and it can handle specific particle hypotheses. Thus, complicated cascade decay patterns, such as the decay of B-mesons into final states including intermediate D-mesons are easily dealt with. It is also rather simple to include several final states in the same analysis. This is very suited for a specific decay mode containing an intermediate resonance like the D's, which decays to many final states with small branching ratios. The physicist actually can spend more time on physics issues rather than solving technical problems. This allows even a newcomer to be productive rather quickly.

In this user manual we consider frequently examples from charm and beauty physics. It should be emphasized, however, that KAL can be used for **any** analysis. It is most useful for analyses containing multiparticle final states or cascade decay chains.

2.1 Example: B-Physics

B-reconstruction is complicated by the fact that many final states exist. In addition many channels contain charmed mesons which again decay to many different final states. Thus, one has several problems:

- a huge combinatorial background
- excellent book keeping needed
- potential double counting

The background can be minimized by:

- reconstructing charmed mesons
- use of particle identification
- applying mass constrained fits to intermediate states
- using sophisticated selection criteria

With FORTRAN this is a tedious task.

2.2 Requirements

The requirements such a general data analysis tool should meet are:

- coding should be close to physics, i.e. all book keeping should be implemented
- easy to use, but should not be restricted to individual analyses
- take care of combinatorics
- make use of particle identification
- allow kinematic and vertex fitting
- possibility to exclude some particles from parts of the analysis
- do Lorentz transformations
- flexible in use and expandable
- access to Monte Carlo information
- easy interface to PAW [2] to store information in histograms and ntuples

2.3 KAL - Basic ideas

KAL was developed by Hartwig Albrecht during the start of ARGUS with these requirements in mind. Bernard Spaan (in the CLEO experiment at Cornell) made KAL modular so that it could be easily implemented into other experiment's software packages, such as CLEO's [3] and ZEUS' [4]. Thus, it was no problem to implement it into DELPHI. KAL has special statements to IDENTIFY particles, SELECT particle combinations and SAVE particle combinations (that can be used in other SELECTs). It has an easy interface to PAW so variables can be stored in ntuples (both Column Wise Ntuples and Row Wise Ntuples) and histograms. Most of the information in the DELPHI standard common blocks in **stdcdes.car** filled by SKELANA [5] is available in the DELPHI KAL version. KAL is also interfaced to several useful routines in the DSTANA [6] program library, and is implemented as a subroutine in the usual PHDST [7] program structure.

Although KAL is a language, no compiler is needed. The KAL program is an interpreter, but the execution speed does not suffer from that. The KAL program 'compiles' the statements into a command stack which is executed via ASSIGNED GOTO statements. KAL is implemented in FORTRAN77, but may best be described as a 4'th generation language.

KAL is executed on an event-by-event basis. For each event KAL loads measured tracks from the main DELPHI track array VECP array into the /CM/ buffer. The process of particle identification then creates 4-vectors in the /CK/, or 'kinematic', bank. This bank may contain multiple entries from the same /CM/ track, but the subsequent kinematic analysis of these entries is protected against nonsense combinations of tracks with itself to produce a composite. In addition to this implicit locking of tracks, KAL allows the user to explicitly LOCK out any arbitary track from any part of the analysis. This results in a powerful and safe analysis environment.

3 Running KAL

KAL is a library (called **libkalxx**) in the official DELPHI libraries, so no compiling is necessary. KAL is a SKELANA [5] based program in the way that it uses the information in the standard common blocks of **stdcdes.car** filled by SKELANA. It is also a continuation of the SKELANA philosophy of 'hiding' the complicated data strucutures and making life easier for the end users. KAL follows the usual PHDST program structure so it has a standard job control. The user only needs to add the following lines in hers/his main program (**skelana.car** for example):

- In USER00: CALL KAL00

- In USER01: CALL KAL01

- In USER02: CALL KAL02 (This should be done after: CALL PSBEG)

- In USER99: CALL KAL99

Your usual UNIX script (NB! Not to be confused with the KAL script in the next section) should contain everything needed by PHDST/SKELANA. In addition you must link to the KAL library **-lkalxx** and the KAL common blocks in **kal.car**:

```
#
#...Link to the KAL library -lkalxx in $DELPHI_LIB:
#
set DELLIBS  = ‘‘-ldstanaxx -lskelanaxx -lkalxx -lphdstxx -lvdclapxx
-ldstanaxx -lphdstxx -lvdclapxx -lufieldxx -lpxtag2xx -lpxdstxx’’

#
#...Necessary KAL common blocks in $DELPHI_PAM/kal.car:
#
+USE,P=MYCOMM.
+PAM,11,T=A,C, R=MYCOMM.              (DELPHI_PAM)/kal.car
```

```
+USE,P=KALCOM.
+PAM,11,T=A,C, R=KALCOM.                (DELPHI_PAM)/kal.car
```

The KAL block data is contained in the DELPHI block data **delblkd.o** in $DELPHI_BLKD.

4 The KAL Script

In KAL the user no longer writes the analysis as a FORTRAN program, but instead writes a separate KAL script containing the analysis in the form of statements. The script is read on an event-by-event basis (when calling KAL02 in USER02), and KAL returns job control to USER02 when it reaches the end of the script. The user is free to do whatever she/he wants to do both before and after the call to KAL. The script should be in a separate file and must be called **script.kal** . When running a job it should be copied to the directory where the job is running. This script is then read and interpreted by KAL. A file called **kal.output** will contain error messages and general information about the KAL script. The user is recommended to always look at this file to verify that the KAL script is correct. If KAL discovers an error it will indicate where in the script it occured. Depending on what the user wants to do, KAL will also make the files **kal.ntuple** which contains all the ntuples and histograms, and **kal.evtlist** for a normal PHDST eventlist file.

What follows is an example of a complete script (i.e. a complete analysis) reconstructing a D_s^+ in the mode:

$$D_s^+ \to \phi\ \pi^+ \quad \text{where: } \phi \to K^+K^-$$

Some explanation is given as comments in the KAL script. Further details are given in the following sections. For detailed explanations of CUTs and variables, see appendix B and C. The examples are given to illustrate KAL features only.

The convention chosen in this DELPHI note is that KAL statements are written in capital letters .e.g. IDENTIFY, while KAL variables and parameters are written in boldface e.g. **nevent**. Common blocks in the DELPHI standard commons all start with 'PSC' and are written in capital letters .e.g. PSCBTG. Steering flags in SKELANA are also written in capital letters e.g. IFLVEC. The main track array in DELPHI is written in capital letters and is called VECP. Individual entries in this array are written in boldface e.g. **vecp (4,i)**. Files are written in boldface e.g. **stdcdes.car**, and subroutines are written in capital letters e.g. subroutine EXPVAR.

4.1 Example of a KAL script

```
kal on                          ; The script should always start with: kal on
                                ; ‘‘;’’ means comment in KAL and everything to
                                ; the right of it is ignored.


cut kasigric -5                 ; Disable particle ID for kaons (i.e. all
```

```
                          ; particles will be considered kaons).
                          ; See appendix B for more information.
cut pisigric 1            ; Want only particles tagged by the RICH
                          ; as pions.


identify k+ k+            ; Load charged tracks into KAL under
                          ; the kaon hypothesis (K+ and K-).
identify pi+ pi+          ; Load charged tracks into KAL under
                          ; the pion hypothesis (pi+ and pi-).


select k+ k-              ; Make K+K- combinations.
  save phi dmass 0.025    ; Save combination as a phi if within 25 Mev
                          ; of the nominal phi mass.
endsel                    ; Defines the end of the K+K- loop.


selcc phi pi+             ; Select phi pi+ and phi pi- combinations.
                          ; ``selcc'' means charge conjugated also.
  save ds+                ; Save combination as Ds+ if phi pi+ and
                          ; Ds- if phi pi-.
endsel                    ; End of phi pi+/phi pi- loop.


selcc ds+                 ; Access all Ds+ and Ds-.
  dsp = p                 ; Keep Ds momentum and mass in user
  dsm = mass              ; defined variables ``dsp'' and ``dsm''.

  seldaug phi             ; Access the phi daughter.
    phim  = mass          ; Keep phi mass and
    phipx = px            ; x component of phi momentum.

    seldaug k+            ; Access positive kaon daugher of phi.
      krichtag = khaidika ; Keep the RICH kaon tag.
    endsel                ; End of K+ ``loop''.
  endsel                  ; End of phi ``loop''.

  seldaug pi+             ; Access the pion daugher (pi+ if Ds+,
                          ; pi- if Ds-).
    pivdhit = nasshit     ; Keep number of VD hits for pion.
  endsel                  ; End of pion ``loop''.




                          ; Put everything in an ntuple (yes, KAL
```

```
                          ; calls it an ‘‘ntupel’’.

  ntupel dsp dsm phim phipx krichtag pivdhit @
  text  'Ds information' @
  tags  'dsp dsm phim phipx krichtag pivdhit'

endsel                    ; End of loop over all Ds+/Ds-.



frame mc                  ; Access MC (generator) information.
  selcc ds+               ; Get Ds+ and Ds-.
    dsp = p               ; Keep Ds momentum.

    seldaug phi           ; Access phi daughter.
      phim = mass         ; Keep phi mass.
    endsel                ; End of phi ‘‘loop’’.

    ntupel dsp phim @           ; Put Ds momentum and
    text  'MC information' @    ; phi mass in ntuple.
    tags  'dsp phim'
  endsel                  ; End of loop over all generator Ds+/Ds-.

endframe                  ; Back to FRAME RECO (default) i.e.
                          ; reconstructed tracks.

return                    ; This returns job control to USER02 in PHDST.
```

This simple script contains only a few of KAL's features, but shows how an analysis is written within this framework. In the following more details and explanations about the generic KAL script will be given. It should be added that KAL is case insensitive i.e. both upper- and lower-case can be used in the KAL script. Also note that KAL only knows about real numbers. Thus, in the KAL script the variable **nevent,** containing the event number, is a real number. All integer numbers in the KAL script are automatically transformed into real numbers by KAL i.e. '1' and '1.0' will have the same effect. The supported arithmetic functions are: SQRT SIN COS TAN ASIN ACOS ATAN LOG LOG10 ABS EXP. Also remark that all arithmetic operators MUST be enclosed in blank spaces! This also includes parenthesis. See 'Assignment statements in KAL' in the KAL main www-page for more details.

The character '**;**' in the script means a comment and everything to the right of it will be ignored by KAL.

Some statements can become very long and won't fit into a single line, like an NTUPEL declaration. A special character, the '**@** ' is reserved to indicate that a continuation line will follow. You may provide as many continuation lines as you want as long as the total length of the statement does not exceed 4048 Bytes. Note: the continuation character '@' has to be separated by at least one blank from the last other character in that line.

Otherwise it won't be accepted. In addition, it should be the last character in that line. Empty lines in the KAL script are not necessary, and are used in this note for graphical purposes only.

4.2 Generic KAL script

4.2.1 IDENTIFY - IDENTSH - IDENTV0

The following statements are available:

```
identify e+  e+
identify mu+ mu+
identify pi+ pi+
identify k+  k+
identify proton  proton
```

With these statements measured charged tracks are loaded into KAL and are given a certain particle hypothesis depending on the value of the CUTs. IDENTIFY assigns the following particle identification hypotheses: **electron, muon, pion, kaon** and **proton** to the tracks using the available detector information and particle identification tags (loose, standard and tight) according to the user's CUTs. Two names are required since you may want to identify particles with more than one set of cuts. The first name corresponds to the particle hypothesis known to KAL, the second name corresponds to the name with which the identified particles may be referred to subsequently in the program. Note that the statement: IDENTIFY K+ K+ assigns the K^+ particle hypothesis to positively charged tracks, while negatively charged tracks will automatically be given the K^- hypothesis, so no additional: IDENTIFY K- K- statement is needed. The order in which the tracks are identified does not matter. Note that the same particle can be assigned all five charged hypotheses, but KAL will make sure that the same particle is not combined with itself when making up composite particles.

```
identsh photon photon
```

This statement assigns the photon hypothesis to neutral electromagnetic showers in the VECP array. This means that to get converted photons the SKELANA [5] flag IFLVEC must be set to 2 (or 22). Please note that KAL does not know about SKELANA 'locking' of tracks so IFLVEC = 22 will have the same effect as IFLVEC = 2. By using the different CUTs one may also use this statement to access the hadronic showers. When requirering zero electromagnetic energy and accepting hadronic energy, the statement: IDENTSH photon hcalsh will identify hadronic showers which will be known inside KAL as 'hcalsh'. See appendix B for further details about these CUTs.

```
identsh pi0 pi0
```

A π^0 fitting program has been implemented in KAL, and this statement will automatically fit π^0s from all neutral electromagnetic showers in the VECP array.

```
identv0 pscpi0 pscpi0
```

This statement makes π^0s from the common block PSCPI0 available for the user. The reason these π^0s are treated differently from the ones from IDENTSH is that they are reconstructed by ELEPHANT [8] from single (overlapping) showers in the HPC i.e. they have no daughter photons. Please note that with: IDENTV0 pscpi0 pi0 they will be known inside the KAL script as π^0s (like the ones from IDENTSH). They can then both be accessed by: SELECT pi0. This is not recommended however since the difference in the number of daughters may cause problems.

```
identv0 k0s  k0s
identv0 lam0 lam0
```

These statements make tight (tagging flag 22 and 33) K_s^0 and Λ from the common block PSCRV0 available to the user. To be able to use the V0s it is necessary that they also appear in the VECP array. This means that the SKELANA flag IFLVEC must be set to 2 or 22, or the V0 must be added to the VECP array 'by hand' before calling KAL.

Only the tracks and neutral showers which pass the acceptance and particle identification CUTs are accepted for a particular hypothesis. The value of the CUTs can be changed by the user in the KAL script. Some cuts are common for all five charged hypotheses (momentum, polar angle ...), others are specific for each hypothesis (particle identification, probability of π^0 fit ...).

4.2.2 CUT

Acceptance and particle identification cuts are set with the CUT statement:

- Control the identification of charged particles
- Control the identification of V0's.
- Control the identification of π^0s.
- Control the identification of photons
- Electron identification
- Muon identification
- Pion identification
- Kaon identification
- Proton identification
- Control the filling of the measured particle banks
- Set certain flags to control program execution

They all have default values which will be used unless the user explicitly changes them. The syntax is:

```
CUT <parameter> <value>
```

Example: Cuts on particle identification for kaons.

```
cut kasigric 2
```

means that only particles tagged as standard (or better)[1] kaons by the RICH (HADSIGN) will be assigned the kaon hypothesis in a subsequent: IDENTIFY K+ K+ statement.

```
cut kasigric -n
```

where **-n i**s any negative number will disable the kaon particle identification and all particles will be assigned the kaon hypothesis in a subsequent: IDENTIFY K+ K+ statement.

The default values are such that all charged particles tagged as at least loose (electrons, muons, kaons, pions and protons - the last three by the RICH (HADSIGN)) will be assigned that specific particle hypothesis in an IDENTIFY statement unless the user explicitly changes this with a CUT statement. A single charged track can be assigned all five hypothesis. It is also possible to combine information from several subdetectors to assign a specific particle hypothesis. The WWW-page: **KAL home page** $\longrightarrow$ **Statements** $\longrightarrow$ **Cut** gives a complete list of the CUTs and their default values. They are also listed in Appendix B. Note that except for particle identification, the default values of the CUTs are chosen such that they are always fulfilled e.g. the minimum momentum for a track is 0.0

4.2.3 SELECT - SELCC

This statement makes it possible to do particle combinatorics with particles that have been previously identified. Up to four particles can be selected with a single SELECT statement. Examples:

```
select k+                          ; Loops over all K+ in the event.
  ..... any statement
endsel                             ; Defines the end of the loop.


select k- pi+ pi+ pi-              ; Loops over all K- pi+ pi+ pi-
  ..... any statement              ; combinations.
endsel                             ; Defines the end of the loop.
```

It is also possible to select both the state and the charge conjugate state at the same time. This is done with: SELCC

```
selcc k+                           ; Loop over both K+ and K-.
  ..... any statement
endsel                             ; Defines the end of the loop.


selcc k- pi+ pi+ pi-               ; Loop over all K- pi+ pi+ pi- and
                                   ; K+ pi- pi- pi+ combinations.
 ..... any statement
endsel                             ; Defines the end of the loop.
```

[1]Tagging flag greater or equal to 2.

Please note that if there are more than two particles of the same type (like "... pi+ pi+ ..." and "... pi- pi- ..." in the previous example), they must be declared in subsequent positions.

One should never use the SELCC statement to select a neutral particle (combination) since this will cause it to be counted twice!

```
selcc k+ k-                    ; Will make k+k- AND k-k+ combinations
                               ; (which will be the same).
                               ; Use SELECT instead!
  ... any statement
endsel
```

4.2.4 Variables

Inside the select loop the user has access to PREDEFINED kinematic quantities like mass, momentum, decay angles in different frames etc. See **KAL home page** $\longrightarrow$ **Variables** and Appendix C for a complete list of KAL variables.

For measured tracks all the variables in the DELPHI standard common blocks in **stdcdes.car** filled by SKELANA [5] are available inside KAL. As was seen in the example in 4.1 they have a different name inside KAL since KAL does not use arrays. The general convention is that the KAL name of a variable is the name of the standard common array plus the three last letters in the corresponding pointer. For example will the x-component of the momentum of track number **i**, which is placed in **vecp (1,i) = vecp (iparpxx,i)**, be called **vecppxx** in KAL. The same way, the impact parameter in $R\Phi$ will be **qtracimp.** Note that variable names that are not arrays have the same name in KAL as in the standard commons i.e. **nevent, nvecp ...** are called **nevent, nvecp ...** In the case where there are no pointer (like for **vecp (2,i)** , the usual convention has been 'extrapolated' i.e. the y-component of the momentum is called **vecppyy** . A complete list of what the SKELANA variables are called in KAL are available from the WWW-page: **KAL home page** $\longrightarrow$ **Variable names**

4.2.5 SAVE - SAVEFITM

The user may also SAVE a selected particle combination and use it in a subsequent SELECT statement. Example:

```
select k+ k-              ; Select all K+K- combinations.
  save phi                ; Save the combination as a phi.
endsel                    ; End of loop over all K+K- combinations.

select phi                ; Previous phis are chosen.
  ..... any statement
endsel                    ; End of phi loop.
```

The full syntax of the SAVE statement is:

```
save <particle> dmass x.xxx chi2 y.yyy
```

The SAVE command defines several new variables:

accept	+1 if the particle is saved -1 if it is not saved
dmass	Mass difference = measured mass - mass (particle table)
chi2	(meas mass - mass(part. table))**2 / error**2 (**chi2** is stored, in contrast to **dmass** and **error** , and can be used in subsequent loops)

Here the particle is saved only if the difference between the measured invariant mass and the nominal mass is less than **x.xxx** GeV/c**2 and/or **chi2** is less than **y.yyy**. Both **dmass x.xxx** and **chi2 y.yyy** may be omitted. In that case, no cut on the mass difference and/or **chi2** are applied. A particle which has been SAVEd before (or which is a measured particle) might be SAVEd again. A new particle bank is booked which is an exact copy of the old bank. All statements within the corresponding loop refer to the new particle bank only. The number of measured and saved particles must not exceed 4000.

The nominal mass for the different particles are taken from the particle table which can be found in the common block CPARTLBD in **kal.car**.

It is also possible to make a mass constrained fit of composite particles. The SAVEFITM statement creates a new particle bank in the same way as SAVE, and adjusts the energy and momentum of the particle with the following constraint:

mass = sqrt (E**2 - p**2) = mass (particle table)

The syntax is exactly the same as for the SAVE statement:

```
savefitm <particle> dmass x.xxx chi2 y.yyy
```

Note that the variable **mass** is not affected by the fit. For all following kinematics however, the fitted mass value will be taken into account. The mass contraint fit manipulates the 4-vector of the current particle combination. Quantities like *E, p* ... and the error matrix are redefined after a SAVEFITM. Consequently, any subsequent SAVE will save the altered 'particle'. The **chi2** cut in any subsequent SAVE will be very large since **mass** is unchanged and the error on **mass** is close to 0. Omit **chi2 y.yyy** if you want to SAVE the particle again! Obviously, the statement SAVEFITM K*0 x.xxx does not make any sense as the natural width of the K^{*0} is much greater than the experimental mass resolution.

If the particle is not saved due to the cuts on **dmass** or **chi2**, the fit is not performed. If an error is detected during the fit procedure, the particle is SAVEd with unfitted values for *E, p* ... The **error** flag has the following values (error messages are printed ten times only):

```
-1 : OK
+2 : Inconsistent derivative vector
+3 : Fit diverged
```

```
+4 : Maximum number of iterations reached
+5 : Negative diagonal element in error matrix
```

The same variables that are defined after SAVE are defined after SAVEFITM.

The example:

```
select k+ k-
  pold = p
  savefitm phi dmass 0.009
  if accept = 1 then
    show p pold
  endif
endsel
```

might yield:

```
    P            POLD
   0.49995      0.50192
```

4.2.6 FITE

The FITE statement is an energy constraint fit for composite particles. It adjusts the invariant mass and momentum of the composite particle with the following constraint:

```
E = sqrt (Mass**2 + P**2)
```

The syntax is:

```
FITE <energy-constraint>
```

Error messages are printed 10 times only, and the **error** flag can have the following values:

```
-1 : OK
 5 : Negative diagonal element in error matrix
```

The variables **chi2** and **error** are defined after FITE.

4.2.7 SELNTHD - SELDAUG - SELDCC

For a selected system, like a ϕ made from K^+K^-, one may want to access the daughter particles and their properties. This is done by the statement: SELNTHD n

The complete syntax is:

```
selnthd <n>
  ... any statements ...
endsel
```

where **n** refers to daughter particle number **n** in the sequence in which the particles have been defined in a SELECT statement: 1 ... **n** ... 4. The statements between SELNTHD and ENDSEL form a loop which is executed exactly once. Nested and/or subsequent loops give access to grand-daugthers, sisters, and so on. Example:

```
select k+ k-              ; Select K+ K- combinations.
  save phi                ; Save them as a phi.
endsel                    ; End of loop over K+ K- combinations.

select phi                ; Select the previous phis.
  selnthd 1               ; Select the first daughter i.e. K+.
    .... any statement
  endsel                  ; Back to the mother (phi).

  selnthd 2               ; Select the second daughter i.e. K-.
    .... any statement
  endsel                  ; Back to the mother (phi).

  selnthd 3               ; Severe error!
    ....                  ; The phi has only 2 daughters!
  endsel                  ; A warning message will be printed.

endsel                    ; End of loop over phis.
```

A second possibility is to access the daughters by name: SELDAUG 'particle name'

```
selcc k- pi+ pi+          ; Select both K- pi+ pi+ and K+ pi- pi-
                          ; combinations .
  save D+                 ; Saved as D+/D-.
endsel

selcc D+                  ; Get all D+/D-.
  seldaug pi+             ; Access all pi+ daughters ... BUT:
    .....                 ; If D- is selected, pi- is selected.
  endsel                  ; Back to mother.
endsel                    ; End of loop over D+/D-.
```

SELDAUG gives access to all of the specified particles which are daughters of the selected system. If the charge conjugate state of the mother is selected, the corresponding anti-particle will be accessed. In the previous example will: SELDAUG pi+ select the π^+ daughter if we are looking at a D^+, but it will select the π^- daughter if we are looking at a D^-.

A third possibility is: SELDCC 'particle name'. SELDCC is very similar to SELDAUG but considers the charge conjugate state of the daughter as well:

```
select k+ k-        ; Select k+k- combinations.
```

```
  save phi          ; Save them as phis.
endsel              ; End of loop over all k+k- combinations.

select phi          ; Select previous made phis.
  seldcc k+         ; Choose both K+ and K- daughters.
    ....
  endsel            ; End of loop over kaon daughters.
endsel              ; End of loop over phis.


select phi          ; Select previous made phis.
  selnthd 1         ; Select K+ daugther of phi.
    ...             ; Remember, the phi is made of K+ K-
                    ; i.e. K+ first.
  endsel            ; End of ``loop'' over K+ daughter.

  selnthd 2         ; Select K- daugther of phi.
    ...
  endsel            ; End of ``loop'' over K- daughter.

  seldaug k+        ; Select K+ daugther of phi.
    ...
  endsel            ; End of ``loop'' over K+ daughter.


  seldaug k-        ; Select K- daugther of phi.
    ...
  endsel            ; End of ``loop'' over K- daughter.

  seldcc k+         ; Select both K+ and K- daughters.
    ...
  endsel            ; End of loop over kaon daughters.
endsel
```

To get access to grand-daughters one simply uses the statement twice ie:

```
select k+ k-            ; Select K+ K- combinations.
  save phi              ; Save them as phis.
endsel                  ; End of loop over K+K- combinations.

selcc phi pi+           ; Select phi pi+ and phi pi- combinations.
                        ; ``cc'' means charge conjugated.
  save ds+              ; Save them as Ds+ and Ds- respectively.
endsel                  ; End of loop over phi pi+ and phi pi- combinations.

selcc ds+               ; Select Ds+ and Ds-.
  selnthd 1             ; Access the first daughter of the Ds i.e. phi.
```

```
    selnthd 1           ; Access the first daughter of the phi i.e. K+
      ....
    endsel              ; End of ‘‘loop’’ over K+ daughter of phi.
  endsel                ; End of ‘‘loop’’ over phi daughter of Ds.

  seldaugh phi          ; Access phi daughter of Ds.
    seldaug k+          ; Access kaon daugher of phi.
     ....
    endsel              ; End of ‘‘loop’’ over kaon daughter of phi.
  endsel                ; End of ‘‘loop’’ over phi daughter of Ds.
endsel                  ; End of loop over Ds.
```

Note: SELNHTD n is the recommended and safest way to get access to daughter particles.

4.2.8 User defined variables

The user is free to define hers/his own variables in the KAL script. They may be up to eight characters long and the name must be different from the PREDEFINED variables (that is **mass, p, px** etc. See appendix C for a complete list). They should not contain '( )' or ','. Numbers and the characters * + - /are not allowed in first position. Nine hundred variables can be defined. This enables the user to keep information that can be used at a later stage in the KAL script. PREDEFINED variables are always defined with respect to the last SELECT statement which means that if there are several SELECT statements, the variables must be stored temporarily until they are put into an ntuple or histogram (or no longer are necessary to keep).

```
select k+ k-             ; Make phis from K+K- combinations.
  save phi
endsel

select phi               ; Here mass, p, px etc are those of the phi,
  seldaug k+             ; but here mass, p, px etc are those of the K+.
    ...
  endsel
endsel
```

A solution to this problem is:

```
select k+ k-             ; Make phis from K+K- combinations.
  save phi
endsel

select phi               ; Select the previous made phis.
  phim = mass            ; Keep phi mass in a new user
                         ; defined variable 'phim'.
  phip = p               ; The same for the phi momentum.
```

```
  seldaug k+              ; Now mass, p etc are those of the K+.
    km = mass             ; Keep the kaon mass and kaon momentum
    kp = p                ; in new variables 'km' and 'kp'.
  endsel

  ..... any statement     ; Can now use both phim, phip, km, kp.
                          ; We will soon see how to make Ntuples.
endsel
```

4.2.9 Interface to PAW

KAL has a very easy interface to PAW [2] enabling variables to be stored both in histograms, Row Wise Ntuples and Column Wise ntuples.

Histograms are defined as they are needed:

```
select k+ k-              ; Make phis from K+K- combinations.
  save phi
endsel

select phi                ; Select the previous made phis.
                          ; Put the phi mass in a histogram.

  plot mass low 0.9 high 1.3 nbin 100 text 'Phi mass'
endsel
```

Before processing the first event, a 1-dimensional HBOOK histogram will be booked defining a lower bound of 0.9, and an upper bound of 1.3, with a total of 100 bins. During event processing, the histogram will be filled with the mass of each individual ϕ.

The complete syntax is:

```
PLOT <var> LOW nn HIGH nn BINSIZE nn NBINS nn @ (cont'd)
           TEXT 'any text' WEIGHT <expr>
or
PLOT <var> FROM nn TO nn STEP nn @ (cont'd)
           TEXT 'any text' WEIGHT <expr>
```

where:

var is the name of the variable to be histogrammed.

low nn or **from nn** defines the lower edge of the histogram. If the lower edge is 0, 'low 0' can be omitted.

high nn or **to nn** defines the higher edge of the histogram.

binsize nn or **step nn** defines the binsize.

nbins nn defines the number of bins. If the number of bins is 120, 'nbins 120' can be omitted.

text 'any text' histogram description (including blanks, numbers, and lower case characters).

weight weight for histogrammed variable.

The binning definition has to be sufficient. If too many or insufficient parameters are given, the program will terminate.

Ordinary Row Wise Ntuples are defined as they are needed:

```
select k+ k-           ; Make phis from K+K- combinations.
  save phi
endsel

select phi             ; Select the previous made phis.
  phim = mass          ; Keep phi mass and momentum in new
  phip = p             ; user defined variables ‘‘phim’’
                       ; and ‘‘phip’’.

  seldaug k+           ; Select K+ daughter of phi.
    kp = p             ; Keep momentum and RICH tag of K+ in
    krich = khaidika   ; user defined variables ‘‘kp’’ and ‘‘krich’’.
  endsel               ; End of ‘‘loop’’ over K+.

                       ; Put everything in an ntuple.
  ntupel phim phip kp krich @      ; @ is cont’d telling
  text  'This is an ntuple' @      ; KAL the line continues.
  tags  'phim phip kp krich'

                       ; And only kaon information in a second ntuple.
  ntupel kp krich @
  text  'Kaon information only' @
  tags  'kp krich'
endsel
```

The complete syntax is:

```
ntupel  <var1> <var2> <var3> ... <var295> @ (cont'd)
```

```
text   'any text' @
tags   'tags'
```

where:

var1 ... var295 are the names of the variables to be histogrammed. The minimum number of variables is two, the maximum is 295.

text 'any text' defines the text for the Ntupel definition.

tags 'tags' defines tags for the Ntupel definition. Tags have to be separated by blanks and must be at most eight character long. The number of tags given has to match the number of variables. If that is not the case, default tags named **'var1'.....'var295'** will be used.

Column Wise Ntuples are also available in KAL [2] Compared to ordinary RWN there are some changes. Your CWN has to be booked at the beginning at the KAL script (unlike the ordinary RWN that is booked automatically). Each CWN has to be given an identification number and a name. The syntax is:

```
CWNBOOK ID nn TEXT '.......'
```

Example: CWNBOOK ID 10 TEXT 'Booking a CWN'

The maximum number of CWN is 10.

Filling blocks in the CWN is done with:

```
CWNTUP var1 var2 ..... varn @
ID nn  BLKNAM 'Block_name' BLKFILL nn TAGS 'Your tags'
```

where:

var1 var2 ... var50 are the variables to be stored in the block. Maximum 50 variables can be stored in one block.

id nn is the ID number of the CWN (ie 10 with the booking over)

blknam 'Block name' is the block name. The name must be unique and no more than eight characters long. Maximum number of blocks in one CWN is 50.

blkfill nn is the maximum number of times the block will be filled in this event.

tags 'Your tags' are the tags for the CWN. They should be a maximum of eight characters long, separated by a blank and their number must match the number of variables.

[2]Thanks to Gabriele Zacek in the ZEUS experiment ad DESY. This is a nice example of a detector independent feature of KAL being copied from one experiment to another.

Unlike ordinary RWN there are no default tags.

Example:

```
CWNTUP mass p px py pz @
ID 10 BLKNAM 'Trk_info' BLKFILL 5 TAGS 'm ptotal px py pz'
```

The maximum number of words to be stored for all blocks in one CWN should not exceed 7000 words per event. An error message will appear in **kal.output** if this limit is reached. For blocks with variable length (**blkfill** greater than one) the running indices are called **lvar1** for the first block with variable length, **lvar2** for the second block with variable length, etc.

Please note that the lenght of the buffer is 4096 i.e. in PAW, to connect the ntuple produced by KAL you need to type: **h/file 1 kal.ntuple 4096**

4.2.10 LOCKING

To analyse parts of an event, some of the particles may be excluded from parts of the analysis. This is done with:

LOCK: Locks all particles which have been selected by the preceding SELECT statement. These particles cannot be selected by any following SELECT statement.

LOCK OTHERS: Locks all particles which have NOT been selected by the preceding SELECT statement. In following SELECT statements, only parts of the previously defined system can be selected.

UNLOCK: Unlock all particles.

Particles are LOCKed only if they are LOCKed before the beginning of a loop i.e. LOCK is not active inside the loop where LOCK is defined. LOCK is active after the end of the loop and inside all nested loops. The LOCK statements selects certain parts of the event for the following analysis. It must be preceded by a SELECT statement and acts on the following SELECT statements.

Note that the LOCK algorithm does not care about mass hypotheses. The only criterion is the track number. Thus, if e.g. a particular proton is LOCKed, all other accepted hypotheses (electron, muon, pion, kaon) of the same track are automatically LOCKed. A composite particle (e.g. a resonance) is LOCKed as soon as one of its contributing measured particles is LOCKed. If a composite particle is LOCKed, all contributing particles are automatically LOCKed. Beware of the fact that LOCK does not prevent a FRAME RECO particle from being MATCHed! (see section 5.1 about Monte Carlo).

The LOCKing feature is also used in connection with the vertex routines (see section 5.3 about Vertex fitting in KAL). Example: Fit a secondary vertex

```
select k+ k-        ; Loop over K+K- combinations.
  save phi          ; Save them as phis.
endsel              ; End of loop over K+K- combinations.
```

```
selcc phi pi+       ; Loop over phi pi+ and phi pi- combinations.
  save ds+          ; Save them as Ds+ and Ds- respectively.
endsel              ; End of loop over phi pi combinations.

selcc ds+           ; Select previous made Ds+ and Ds-.
  lock others       ; Lock tracks that are NOT used to make the Ds+/Ds-.
    fitall          ; Fit a secondary vertex of tracks
                    ; contributing to the Ds only (ie K+ K- pion).
  unlock            ; Unlock tracks.
endsel              ; End of loop over Ds+/Ds-.
```

4.2.11 FRAMEs

The FRAME statement allows to switch from one frame of particles to another one:

FRAME RECO: All reconstructed (measured and composite) particles pointing to the main vertex in the rest frame of the DELPHI detector.

FRAME MC: All Monte Carlo generator particles.

Event processing starts in the FRAME RECO, i.e. this statement is not necessary at the beginning of a program. Each FRAME has to be terminated by a corresponding ENDFRAME statement. After ENDFRAME, the program returns to the previous frame. The next section will show how this is done to access Monte Carlo generator information.

5 KAL features

5.1 Monte Carlo

In KAL one also has access to Monte Carlo generator information. This is done in the same way as for the measured tracks, but one must switch to the Monte Carlo frame. The usual frame with measured tracks/reconstructed particles is called FRAME RECO and is the default frame i.e. it does not need to be specified.

```
                    ; Default frame is FRAME RECO
                    ; i.e. measured tracks.

frame mc            ; Switching frame to get MC
                    ; generator information.
  selcc ds+         ; Select generated Ds+ and Ds-.
   ... any statement
  endsel            ; End of loop over generated Ds+/Ds-.
endframe            ; End of MC generator information.

                    ; We are now back to measured tracks i.e.
```

```
; FRAME RECO.
```

All PREDEFINED KAL variables that are available for measured tracks (like mass, momentum, decay angles, ...) are also available for the Monte Carlo tracks. In addition the following variables are defined:

mcx - mcy - mcz: Starting point of particle (ie production vertex).

mcveclin: Contains the VECP track number of the Monte Carlo particle if there is a link. This variable is put to 0 if there is no link.

For the measured tracks the following Monte Carlo generator information is available when there is a link between the measured track and a generated particle. If there is no link, the variables will be put to 0 (**kp1-kp5**) and -99 (**pp1-pp5, vp1-vp5**) (see also the common blocks PSCLUJ and PSCTBL for further information).

kp1 : Status code of current particle
kp2 : Particle code (Jetset convention)
kp3 : Index of mother particle
kp4 : Index of first daughter particle
kp5 : Index of last daughter particle

pp1 : Generated Px in GeV
pp2 : Generated Py in GeV
pp3 : Generated Pz in GeV
pp4 : Generated energy in GeV
pp5 : Generated mass in GeV

vp1 : X-position of production vertex in cm
vp2 : Y-position of production vertex in cm
vp3 : Z-position of production vertex in cm
vp4 : Time of production (currently empty)
vp5 : Proper lifetime of the particle (currently empty)

kp2moth : Particle code of mother (JETSET convention)
indmoth : Track index in VECP array of mother
pp1moth : Generated Px of mother
pp2moth : Generated Py of mother
pp3moth : Generated Pz of mother
pp5moth : Generated mass of mother
kp2gramo : Particle code of grandmother
indgramo : Track index in VECP array of grandmother
kp2ggrmo : Particle code of grandmother's mother
indgrmo : Track index in VECP array of grandmother's mother

```
select k+k-             ; Loop over K+K- combinations.
  save phi              ; Save them as phis.
endsel

select phi              ; Loop over all previously made phis.
  selnthd 1             ; Access first daughter i.e. K+.
    k1code  = kp2       ; Particle code of track if linked.
    k1moth  = kp2moth   ; Particle code of mother.
    k1indmo = kp3       ; Index of mother.
  endsel

  selnthd 2             ; Access second daughter i.e. K-.
    k2code = kp2        ; Particle code of track if linked.
    k2moth = kp2moth    ; Particle code of mother.
    k2indmo = kp3       ; Index of mother.
 endsel

  if abs ( k1code ) = 321 and abs ( k2code ) = 321 then
    if k1indmo = k2indmo then
      if k1moth = 333 and k2moth = 333 then

      .......           ; Here both kaons have been linked to
                        ; generated kaons, and in addition they
                        ; both come from the same mother which is
                        ; a phi.
      endif
    endif               ; Close all IF statements.
  endif
endsel                  ; End of loop over phis.
```

One additional feature is an automatic MATCHING of Monte Carlo generated particels/tracks with reconstructed particles/tracks which is completely independent of the links in the previous example.

```
select k+ k-                    ; Make phi of measured tracks.
  save phi                      ; (need something to match with).
endsel


frame mc                        ; Access the MC information.
  select phi                    ; Get phi.
    match reco                  ; Match the MC phi with any
                                ; reconstructed particle.
      ntupel mass @             ; Put matched reconstructed particle
      text  'Matched phi' @     ; mass in an ntuple (here we are in
      tags  'phimass'           ; FRAME RECO).
```

```
    endmatch                    ; End of matching and
                                ; back to MC (FRAME MC).
  endsel
endframe                        ; Back to RECO frame i.e.
                                ; measured tracks.
```

In this case any reconstructed particle was MATCHed to the Monte Carlo phi. If one wants to make sure that the MATCHed reconstructed particle really is a reconstructed ϕ one can do:

```
select k+ k-                    ; Make phis.
  save phi
endsel

frame mc                        ; Access MC generator information.
  select phi                    ; Select generator phis.
    match reco                  ; Match MC phi with any reconstructed
                                ; particle/track.
      if ident(phi) = 1 then    ; ident(phi) = 1 if the matched particle
                                ; really is a reconstructed phi
                                ; i.e. here we are in FRAME RECO.
        ntupel mass @
        text  'phi is matched' @
        tags  'phimass'
      endif                     ; Close IF statement.

    endmatch                    ; End of matching and back to MC.
  endsel
endframe
```

In the first example KAL would take a Monte Carlo ϕ and MATCH it with all possible measured and reconstructed particles. In the second case, we used the SPECIAL FUNCTION IDENT (particle name) to find out if the MATCHed particle really was one of the previous reconstructed ϕs. Note also that KAL permits the use of IF-THEN-ELSE statements.

5.2 User defined flag

The usual PREDEFINED KAL variables **mass, p, px, ...** are always defined with respect to the last SELECT statement. There is also another sort of variable called **flag** which enables the user to tag a specific particle. For example, in the previous example we MATCHed Monte Carlo ϕs with reconstruced ϕs. It could be useful to tag these reconstructed ϕs so that when we later loop over them, we will know the ones that were MATCHed. In KAL this is done with the following:

```
select k+ k-                    ; Make phis of measured tracks.
  save phi
```

```
endsel

frame mc                          ; Switch to MC frame.
  select phi                      ; Loop over MC phis.
    match reco                    ; Match MC with any reconstructed
                                  ; particle/track.
      if ident(phi) = 1 then      ; ident(phi) = 1 if the matched particle
                                  ; really is a reconstructed phi
                                  ; i.e. here we are in FRAME RECO.
        set flag = 1              ; Mark each of these matched reconstructed
                                  ; phis with flag = 1 (default = 0).
      endif                       ; Close if statement.

    endmatch                      ; End of matching and back to MC.
  endsel
endframe                          ; Back to FRAME RECO.

select phi                        ; Loop over reconstructed phis
  if flag = 1 then                ; If matched ....
    ntupel mass p nevent @
    text  'Matched reco phi' @
    tags  'mass p nevent'
  else                            ; If not matched (flag = 0).
    ntupel mass p nevent @
    text  'Unmatched reco phi' @
    tags  'mass p nevent'
  endif
endsel
```

The variable **flag** is by default put to 0. By using SET FLAG = n inside a SELECT loop we can put a specific value of **flag** to each of the selected particles. Please note that the variable is redefined with the SET statement. The variables **userva01 - userva10** can be used in the same way as **flag.** Please remark that when a photon is used to make a Pi0 it is considered a "new" particle and the previously set user flags will not follow the photon. They can however be set independently for the photon considered as a photon, and for the photon considered as a π^0 daughter.

5.3 Vertex fitting in KAL

KAL also has an interface to several useful routines in the DSTANA [6] program library. In particular it permits vertex fitting (both primary and secondary) in an especially easy way. These routines have a certain number of input parameters like chisquare and impact parameter cut values, use beam spot in fit etc. In general the default values given in the DSTANA writeup are used. These can easily be changed by the user.

To make a **primary vertex fit** with default parameters one only needs to write:

```
primvtx
```

The full syntax is:

```
primvtx ndim n ch2trk n beamsp n nvd n nvdxy n nvdz n ipxy1 n
        ipz1 n ipxy2 n ipz2 n flag n
```

where **n** is the parameter value and:

Parameter:	Explanation:	Default:
ndim	Mumber of dimensions of the fit	3.00
ch2trk	Maximum chi2 contribution pr track	5.00
beamsp	Use beam spot constraint (default) or not (1)	
nvd	Minimium number of VD hits for tracks to be used in fit	0.00
nvdxy	Minimium number of VD hits in RPhi for tracks to be used in fit	0.00
nvdz	Minimium number of VD hits in Z for tracks to be used in fit	0.00
ipxy1	Maximum impact parameter in RPhi for first stage	0.25
ipz1	Maximum impact parameter in Z for first stage	5.00
ipxy2	Maximum impact parameter in RPhi for second stage	0.05
ipz2	Maximum impact parameter in Z for second stage	1.00
flag	Only use flagged tracks: set flag = n (n different from 0)	0.00

KAL will then use the DSTANA routine PNPVFIT to make a primary vertex fit. The result will then be available in the following variables (see the DSTANA writeup for more details):

npviter	Number of iterations used in PNPVFIT
pvierr	Error flag from PNPVFIT: 0 means fit OK
pvntk	Number of tracks used in fit
pvndf	Number of degrees of freedom of fit
pvx	X-coordinate of primary vertex (cm)
pvy	Y-coordinate of primary vertex (cm)
pvz	Z-coordinate of primary vertex (cm)
pvch2	Chisquare of vertex fit
pverr1	Error
....	
pverr6	matrix

Which charged tracks to use in the fit is decided by the user by changing the default parameters. If one would like to use only tracks with at least one VD hit (RPHI or Z) in the fit one would simply write:

```
primvtx nvd 1
```

One can also explicitly flag the tracks one wants to use in the vertex fit. If, for example, one wants to find the primary vertex separately in the two hemispheres (with respect to the btag thrust axis from the common block PSCBTG) one can do:

```
cut pisigric -5          ; Disable particle ID since we want to use
                         ; all tracks for the vertex fit. Now all
                         ; tracks are considered pions.

identify pi+ pi+

selcc pi+                ; Loop over all tracks.
                         ; Find sign of cosine of the angle between
                         ; the track and the thrust axis.
  angthrub = ( ( qbtthr1 * px ) + ( qbtthr2 * py ) + @
               ( qbtthr3 * pz) )

  if 0 <= angthrub then
    set flag = 1         ; Hemisphere 1 wrt the btag thrust axis
  else
    set flag = 2         ; Hemisphere 2 wrt the btag thrust axis
  endif
endsel

primvtx flag 1           ; Here only tracks in hemisphere 1
                         ; wrt the btag thrust axis are used.
pvh1x   = pvxxx          ; Keep PV position and error in
pvh1y   = pvyyy          ; new variables.
pvh1z   = pvzzz
pvh1err = pvierr

primvtx flag 2           ; Here only tracks in hemisphere 2
                         ; wrt the btag thrust axis are used.
pvh2x   = pvxxx          ; Keep PV position and error in
pvh2y   = pvyyy          ; new variables.
pvh2z   = pvzzz
pvh2err = pvierr

ntupel pvh1x pvh1y pvh1z pvh1err pvh2x pvh2y pvh2z pvh2err @
text  'PV_H1/2' @
tags  'pvh1x pvh1y pvh1z pvh1err pvh2x pvh2y pvh2z pvh2err'
```

One also has the possibility to exclude tracks from the fit which one knows are not coming from the primary vertex (like tracks from B and D mesons). This is done with:

```
VETO2PV
```

which makes a list of all UNLOCKed tracks. The tracks in this list will not be used in a subsequent primary vertex fit. Note that a second VETO2PV will simply add the new tracks to the list, it will not overwrite the old ones. PRIMVTX will however overwrite all tracks in the list after the primary vertex fit. One realistic example would be:

```
select k+ k-                    ; Loop over K+K- combinations.
  save phi                      ; Save them as phis.
endsel                          ; End of loop over K+K- combinations.

selcc phi pi+                   ; Loop over phi pi+ and phi pi-
                                ; combinations.
  save ds+                      ; Save them as Ds+ and Ds- respectively.
endsel                          ; End of loop over phi pi combinations.

primvtx nvdxy 1 nvdz 1          ; Make PV with tracks having at
                                ; least one VD RPHI and one VDZ hit.

ntupel pvxxx pvyyy pvzzz pvierr @    ; Store it in an ntuple.
text  'PV - All VD tracks' @
tags  'pvx pvy pvz pverror'

selcc ds+                       ; Select Ds+ and Ds-.
  lock others                   ; Lock all other tracks.
    veto2pv                     ; Don't want to use Ds tracks
                                ; in the PV fit.
    primvtx nvdxy 1 nvdz 1      ; Make PV fit without Ds tracks.

    ntupel pvxxx pvyyy pvzzz pvierr @    ; Store it in an ntuple.
    text  'PV - No Ds tracks' @
    tags  'pvx pvy pvz pverror'
  unlock
endsel
```

KAL also enables you to get the impact parameters wrt the new primary vertex. This is done on a track-by-track basis with the command:

```
NEWIP
```

The new impact parameters will then be available in:

newimpxy	Signed distance (d.c.a on xy projection)
newimpz	Z of perigee
newimpdi	Total distance of perigee computed from **newipxy** and **newipz**
newimpe1	Error matrix on **newipxy** and **newipz** accounting for errors
-	on the vertex and the track itself.
newimpe4	Error on **newipxy** = sqrt (**newimpe1**)
	Error on **newipz rm = sqrt (newimpe3**)

```
Example:  primvtx                    <--- Must have been called once
```

```
selcc e+
   if pvierr = 0 then        <--- If PV fit OK then
     newip

     iprphi = newimpxy         <--- To keep the new IP
     ipz    = newimpz
     ipdist = newimpdi

     iperror1 = newimpe1       <--- Error matrix
     iperror2 = newimpe2
     iperror3 = newimpe3
     iperror4 = newimpe4
   endif
 endsel
```

To make secondary verteces there are three possibilities. FITALL will fit all unlocked charged measured tracks (from PSCVEC) and V0s (from PSCRV0) to a common vertex using the DSTANA routine VDF2ND. The full syntax is:

```
FITALL x n y n z n beamsp n maxniter n veto1 n veto2 n veto3 n
       veto4 n veto5 n
```

Parameter:	Explanation:	Default:
x y z	Initial guess for vertex position	(0,0,0)
beamsp	Use beam spot constraint (1) or not (default)	
maxniter	Maximum number of iterations	50
veto1	May exclude five measured tracks from the vertex fit	
- **veto5**	(use VECP tracknumber i.e. variable **trackno**)	

After the fit the following variables are defined:

error	Actual error from VDF2ND
accept	+1 if fit OK and -1 if fit ended with an error
allvxx	X-position of fitted vertex (cm)
allvxy	Y-position of fitted vertex (cm)
allvxz	Z-position of fitted vertex (cm)
chi2fa	Chisquare of fit
prchi2fa	Chisquare probability of fit
testchi2	Contribution to the total chisquare from the worst track
testrkm	Track number (CM) of worst track
result	Number of tracks (temporary variable!)
vtx2co1	Covariance matrix on
...	x-, y- and z-position of

vtx2co6 the fitted vertex

An example would be:

```
selcc ds+                          ; Select Ds+/Ds- previously made.
  lock others                      ; Lock all other tracks.
    fitall                         ; Fit Ds tracks to a common vertex.
    if accept = 1 then             ; If the fit was successfull:
      dsvx   = allvxx              ; Keep the information.
      dschi2 = chi2fa
    endif
  unlock                           ; Unlock tracks.
  .....
  ntupel accept dsvx dschi2 ... @  ; Put information in ntuple.
  text  'Ds information' @
  tags  'accept dsvx dschi2 ...'
endsel
```

FITVTX is a routine which fits all flagged (by the user) tracks. The full syntax is:

```
FITVTX x nx y ny z nz tag n
```

where **x**, **y** and **z** are the same as for FITALL. **tag n** is the value set with:

```
set flag n
```

The same variables which are defined after FITALL are also defined after FITVTX with the exception of the refitted track parameters and momenta.

To fit a vertex of all tracks in one hemisphere (with respect to the btag thrust axis in the common block PSCRV0) one can do:

```
selcc pi+
  angthrub = ( ( qbtthr1 * px ) + ( qbtthr2 * py ) + @
               ( qbtthr3 * pz) )

  if 0 <= angthrub then
    set flag = 1   ; Hemisphere 1 wrt the btag thrust axis
  else
    set flag = 2   ; Hemisphere 2 wrt the btag thrust axis
  endif
endsel

fitvtx tag 1  ; Fit all tracks in hemisphere 1 to a vertex
```

The third possibility - useful when charmed mesons are involved - is to first reconstruct and fit a vertex for one particle, make a pseudo-track for that particle, and then fit this pseudo-track with some other measured tracks. This can be done with: MAKE2VTX

The full syntax:

```
MAKE2VTX pa1 n pa2 n pa3 n pa4 n x n y n z n beamsp n maxniter n
         veto1 n veto2 n veto3 n veto4 n
```

The new parameters compared to FITALL are **pa1 - pa4** . These are the CK track-numbers (ie variable **tracknok)** of the pseudoparticles to first reconstruct. The output variables are the same as for FITALL. An example:

```
selcc b0                  ; Look at B0 --> D- pi+/D+ pi-.
  seldcc d-               ; Access D-/D+ daughter.
    dtrk = tracknok       ; Get /CK/ track number of D-/D+.
  endsel

  lock others             ; Lock all other tracks than those
                          ; used to recontruct the B0.
    make2vtx pa1 dtrk     ; Tell KAL to first fit a D-/D+ vertex
                          ; and then fit this pseudo-track with the
                          ; pion to a common vertex.
    if accept = 1 then    ; If successfull fit then
      b0vx = allvxx       ; keep vertex position.
      b0vy = allvxy
      .....
    endif
  unlock                  ; Unlock tracks.
  .....                   ; May continue analysis on B0.
endsel
```

With both of these routines the fitted parameters θ, ϕ and κ with their error matrix, in addition to the fitted momenta, are available for each track:

- **thetanew**
- **phinew**
- **kappanew**
- **parerr1 - parerr6**
- **vpxfit, vpyfit, vpzfit**
- **vpfit**

5.4 FORTRAN interface

It may happen that the user wants to do something that is not possible to do with the KAL features currently available. She/he then has the possibility to call a user defined

FORTRAN Subroutine from the script where she/he can write the necessary code. An array **userv(20)** in the common block **mykalcom** may then be filled with the results, and the corresponding variables will then be available inside KAL as **userv1 .... userv20.**

This statement may be called from anywhere in the KAL script. The syntax is:

```
CALLFORT <var1> <var2> ... <var12>
```

The CALLFORT statement has 0 ... 12 parameters which must be variables (i.e. no numbers, no expressions). It calls a FORTRAN subroutine **aukal** which has to be supplied by the user (a dummy version is given in the KAL library, see also **kal.car**):

```
      SUBROUTINE AUKAL (K,NPAR,PAR)
C
C...User routine called by the CALLFORT statement.
C
C   K    :  Particle number in /CK/ (K = 0 if no particle is selected)
C   NPAR :  Number of parameters - PAR (NPAR)
C   PAR  :  Parameters transfered from the KAL script to this subroutine
C           via CALLFORT par1 par2 .... par12

C
C...Common block containing USERV(20):
+CDE,MYKALCOM.

C
C...PAR(1) corresponds to par1 etc:
      DIMENSION PAR (*)

C
C...Your code:


C
C...Put results in array USERV so it can be accessed inside KAL:
      USERV(1) = .....
      .....

C
C...End of AUKAL:
      RETURN
      END
```

After RETURN or END in AUKAL, normal processing of KAL statements continues.

5.5 VD drawings

KAL is interfaced with the DSTANA [6] routine GREASY which draws pictures (postscript files) of tracks and hits in the Vertex detector. Two possibilities exist:

```
drawvda
```

gets the whole VD without any enlargement of the Beam spot.

```
drawvd3
```

gets the whole VD and two enlargements of the Beam spot (in three different ps files).

Note that several calls to these routines in the same event will not have any additional effects i.e. only one (three) ps file(s) will be produced per event.

5.6 Eventlists - Stripping of events

To put the event into a PHDST event list one needs just to write:

```
evtlist
```

in the script. The event list produced by KAL will be named **kal.evtlist.**

It is also possible to strip events i.e. write the event to an output stream defined by the user in the PDLINPUT [7] file. The statement is:

```
WRITE out1 n out2 n ..... out9 n
```

where **n** is the number of the output stream i.e. the same event can be written to nine different output streams at the same time (which is the maximum allowed with PHDST). The variable **nbevtout** contains the total number of events that have been written out and is available in KAL.

5.7 Special functions

These are functions that have particle names as arguments:

```
PMASS   (<id>)  mass of particle <id>
PCHARGE (<id>)  charge of particle <id>
PLIFET  (<id>)  lifetime of particle <id> (sec)
PWIDTH  (<id>)  width of particle <id> (GeV/c**2)

PID   (<id>) number in the particle table of particle <id>
PIDCC (<id>) number in the particle table of particle <id>
             if a charged conjugated system is not selected
             (see SELCC)
                                or
             number in the particle table of the antiparticle
             of <id> if a charged conjugated system has been
             selected
```

If the particle identifier of a selected particle is not known:

```
IDENT   (<id>)  +1 if the selected particle is identical to <id>
                -1 if the selected particle is different from <id>
IDENTCC (<id>)  same as IDENT, but refers to the antiparticle of <id>
                if a charged conjugated system has been selected
IDENTB  (<id>)  +1 if the selected particle is identical to <id> or
                   its antiparticle or
                -1 if the selected particle is different from <id> and
                   its antiparticle
```

Information on mother/daughter relations:

```
MOTHID   (<id>)  +1 if the selected system has <id> as mother
                 -1 if the selected system has no mother or
                    a mother different from <id>
MOTHIDCC (<id>)  same as MOTHID, but refers to the antiparticle of
                 <id> if a charged conjugated system is selected
MOTHIDB (<id>)   +1 if the selected system has <id> or its
                    antiparticle as mother or
                 -1 if the selected system has no mother or one
                    which is different from both <id> and the
                    antiparticle of <id>

NDID   (<id>)   returns how often <id> appears as a direct daughter of
                the selected system
NDIDCC (<id>)   same as NDID, but refers to the antiparticle of <id>
                if a charged conjugated system is selected
NDIDB  (<id>)   returns how often <id> or its antiparticle appear as
                direct daughters of the selected system
```

5.8 New particles

The PARTICLE directive defines new particles in the particle table or redefines properties of particles which are already defined. Syntax:

```
PART <name> ANTI <antiname> MASS <num> CHARGE <num> LIFETIME <num>
```

where: **name** and **antiname** are character strings of up to eight characters. All particles must have their antiparticles (which may be the same as the particle).

For example:

```
PART PI0 ANTI PI0 MASS 100
```

defines a rather heavy π^0 (mass equal to 100 GeV/c**2, while charge and liftime remain unchanged)

In addition, it is possible to define particles which have properties of particles already listed in the particle table:

```
PART <name> alias <name2>  antipart <name>
```

This is especially useful since it allows us to divide measured tracks into two hemispheres. If we want to reconstruct ϕs, but only of kaons in the same hemisphere, we would write (if we use the b-tag thrust axis in the common block PSCBTG to divide the event into two hemispheres):

```
kal on

particle h1k+ alias k+ antipart h1k-   ; Define hemisphere 1
particle h2k+ alias k+ antipart h2k-   ; and hemisphere 2 kaons.

particle h1phi alias phi antipart h1phi
particle h2phi alias phi antipart h2phi

identify k+  k+                        ; Identify kaons.


select k+                              ; Loop over all K+ in the event.

                                       ; Define new variable proportional to
                                       ; cos (kaon-thrust axis)
                                       ; (we only need the sign).
  angthrub = ( ( qbtthr1 * px ) + @    ; cont'd
               ( qbtthr2 * py ) + @    ; cont'd
               ( qbtthr3 * pz ) )

  if 0 <= angthrub then                ; If pointing in same direction
                                       ; as thrust axis, in hemisphere 1.
    save h1k+
  else                                 ; If opposite, hemisphere 2.
    save h2k+
  endif
endsel


select k-                               ; Do the same for K-
  angthrub = ( ( qbtthr1 * px ) + @
               ( qbtthr2 * py ) + @
               ( qbtthr3 * pz ) )

  if 0 <= angthrub then
    save h1k-
  else
    save h2k-
  endif
endsel
```

```
select h1k+ h1k-                    ; Loop over all K+K- combinations
  save h1phi                        ; in hemisphere 1 and save as phi.
endsel


select h2k+ h2k-                    ; Loop over all K+K- combinations
  save h2phi                        ; in hemisphere 2 and save as phi.
endsel

.......

return
```

In this example we define new kinds of particles, h1k+ (with antiparticle h1k-) and h2k+ (with antiparticle h2k-) which have the same properties (ie mass, charge, lifetime) as ordinary kaons (and the same for the ϕs). We then loop over all kaons in the event and check which hemisphere they are in. We then save them accordingly as h1k+/h1k- or h2k+/h2k-. The new 'hemisphere' kaons can then be used in a subsequent SELECT statement to make a new particle (here a h1phi/h2phi). This way the ϕs will not be made up of kaons in two different hemispheres.

6 Future

A first version of KAL has now been available in DELPHI for some time and it is the authors' hope that even more users will discover this useful tool and use it in their analyses. Even though it has been tested some bugs may remain, but we would like to stress that one of the advantages of KAL is that when a bug is corrected it is corrected once and for all. Changing your analysis will NOT introduce new bugs. Any feedback (suggestions, bugs ...) from the users are welcome - just contact: Anders.Borgland@fi.uib.no

Acknowledgements

The authors are greatly indebted to Hartwig Albrecht for having written the original KAL, and especially to Bernard Spaan for having made a large part of the program experiment independent. In addition to making a well defined detector interface, he wrote the original WWW-files for KAL, some of which have been adapted to DELPHI. Parts of these WWW-files have been used in this DELPHI note. He also helped the authors with technical points during their work to adapt KAL to DELPHI.

References

[1] KAL in ARGUS (DESY):
http://www.physik.uni-dortmund.de/ARGUS/Kal/kal.html

[2] PAW - Physics Analysis Workstation:
http://wwwcn1.cern.ch/asd/paw/index.html

[3] KAL in CLEO (Cornell):
http://www.physics.mcgill.ca/WWW/zacek/kalcleo/kal.html

[4] KAL in ZEUS (DESY):
http://www.physics.mcgill.ca/WWW/zacek/ZKAL/intro.html

[5] SKELANA - Skeleton Analysis Program:
http://wwwcn.cern.ch/~cossutti/skelana.html

[6] DSTANA - Analysis Tools Library:
http://infodan.in2p3.fr/delphi/analysis/dstana/dstana.html

[7] PHDST - DST Input/Output Program: DELPHI 92 - 118 PROG 189 Rev 3.
http://delwww.cern.ch:8010/physics/nsmirnov/www/phdst/phdst.html

[8] ELEPHANT - Electron/Photon Analysis Tool: DELPHI 96-82 PROG 217.
http://wwwcn.cern.ch/ pubxx/tasks/elephant/www/Welcome.html

[9] BSAURUS - The DELPHI Inclusive B Physics Program
http://wwwcn.cern.ch/ pubxx/tasks/bcteam/www/inclusive/bsaurus.html

A KAL statements

What follows is a list of all the current KAL statements. Further explanation can be found either in this manual or in the WWW-pages: **KAL homepage** $\longrightarrow$ **Statements**

```
KAL STATEMENTS (Alphabetical)

ADDBUF   : Define event mixing buffer number
AND      : Program control
CALL     : Program control
CALLFORT : FORTRAN interface
CUT      : Set parameters e.g. cuts
CWN      : Column-Wise-Ntuples
DEBUG    : Debugging feature
DEFBUF   : Define event mixing buffer number
DPLOT    : 2 dimensional diagrams
DRAWVD   : Picture of tracks and hits in the Vertex Detector
ELSE     : Program control
ENDIF    : Define the end of IF blocks
ENDPROC  : Program control
ENDSEL   : Define the end of loops
EVTLIST  : Write events to an eventlist file
FILLBUF  : Fill the event mixing buffer
FITALL   : Performs vertex fit on all unlocked particles
FITE     : Fit the energy of composite particles
FITVTX   : Performs vertex fit on all flagged tracks
FRAME /
ENDFRAME : Changing frames
GETBUF   : Get particles from the event mixing buffer
IDENTIFY : Particle ID - charged tracks
IDENTSH  : Particle ID - photons and Pi0s
IDENTV0  : Particle ID - V0s from PSCRV0 and Pi0s from PSCPI0
IF       : Program control
JBIT     : Get a bit
JBYT     : Get a consecutive number of bits
JUMP     : Program control
LABEL    : Program control
LISTCK   : List the kinematics banks
LISTCM   : List the measured particle (CM) buffer
LISTCUTS : List the current CUT parameters
LISTMC   : List the Monte Carlo information
LISTMTCH : List Monte Carlo Data matching
LISTVX   : List the vertex banks
LOCK     : Analyse parts of the event
LORBOOST : Lorentz Transformations
LORDEF   : Lorentz transformations
MATCH /
MATCH1 /
```

```
ENDMATCH : Matching Monte Carlo
NEWIP    : Impact parameters to new PV
MAKE2VTX : Secondary vertex fit with pseudoparticles
NTUPEL or
NTUPLE   : Row-wise ntuples
OR       : Program control
OSELMEAS : Selects only ‘‘measured’’ particles
PLOT     : 1 dimensional histograms
PRIMVTX  : Primary vertex fit
PROC INIT: Pre-analysis calculations
PROC TERM:
RETURN   : Returns to USER02 in PHDST
SAVE     : Save a bank for a new particle
SELCC    : Select a state and its charge conjugate
SELECT   : Selecting one type of particle
SELECT   : Selecting two or more particles
SELMEAS  : Loops over measured particles
SELNTHD  : Access Nth daughter
SELDAUG  : Access daughter particle(s) by name
SELDCC   : Access daughter particles(s) by name + charge conjugate
SET      : Redefine predefined variables
SHOW     : Write variables to the line pr. / terminal
SPECIAL
FUNCTIONS: Particle information
STOP     : Program control
STOREPS  : Manipulate PSEUDO particles.
SUBPROC  : Sub-KAL-program
TABLE    : Fill formatted tables
TABOUT   : Output tables
THEN     : Define IF blocks
VETO2PV  : Tracks to Primary vertex fit
WRITE    : Write events to file
```

B CUT Command

With the CUT command, certain parameters are set which:

* Control the identification of charged particles.
* Control the identification of V0's.
* Control the identification of Pi0's.
* Control the identification of photons.
* Control electron identification
* Control muon identification
* Control pion identification
* Control kaon identification
* Control proton identification

* Control the filling of the measured particle banks
* Control the filling of the variables containing the visible energy
* Set certain flags to control program execution
* Set certain flags to control output / error messages

The syntax is: CUT (parameter name) (num)

The default CUT parameter values will be listed at the beginning of each job (in the file **kal.output**). In order to obtain a listing of the actual CUT parameter values, use the LISTCUTS command.

Cut - parameters for particle identification

The commands listed in this section may be used to modify the particle identification. If they are not given, default values will be used. These default values will be the minimum i.e. a particle is identified if it is tagged as "loose" (or better). Veto and combinations of cuts are possible. Please note that KAL only knows about real numbers, and will automatically transform any integer in the KAL script into a real number. This means that: CUT ELTAG 4 and CUT ELTAG 4.0 will have the same effect. Also note that KAL only "knows about" the tracks that already exist in the VECP array.

```
Charged tracks

The following cuts are valid for all charged tracks independent of particle
hypothesis:

  CUT PMIN       <num> : Defines the minimum momentum
                 0.0   : default

  CUT PMAX       <num> : Defines the maximum momentum
              1000.0   : default

  CUT COSTHETA   <num> : Defines the maxmium absolute value of the
                         cosine of the polar angle
                 1.0   : default

  CUT VOVETO     <num> : Reject or keep tracks used in Tight V0s (from PSCRV0):
                         1 - REJECT
                         0 - KEEP
                 0     : default

  CUT CONVETO    <num> : Reject or keep tracks used in gamma conversions
                         (from PSCPHC):
                         1 - REJECT
```

```
                        0 - KEEP
                0     : default

  CUT NASHTALL   <num> : Defines the minimum number of VD hits.
                0     : default

  CUT CHI2VX     <num> : Maximum chi2 of track fit with VD hits
             1.0E+25  : default

  CUT IPXY       <num> : Maximum impact parameter in XY
              1000.0  : default

  CUT IPZ        <num> : Maximum impact parameter in Z
              1000.0  : default

  CUT TRACKLEN   <num> : Minimum track length in cm
                   0  : default
```

Electron Identification

The following cuts are available for electron identification: (See PSCELO for further explication. Most cuts will always be satisfied i.e. have no effect unless explicitly changed by the user!) Unless stated otherwise, the information is from ELEPHANT.

```
CUT ELTAG    <num>  : Defines the minimum ELEPHANT tag necessary i.e.
                      2 - VERY LOOSE
                      3 - LOOSE
                      4 - STANDARD
                      5 - TIGHT
                   3 : default
                   ex: CUT ELTAG 4 means particles tagged as
                       standard or tight are accepted as electrons
                   ex: CUT ELTAG -5 means all particles will be
                       considered electrons i.e. particle (electron)
                       identification is disabled. Please note that any
                       negative number will have this effect.

CUT ELECNNID <num>  : Defines the minimum value of the the output from
                      C.J. Kreuter's new  Electron Identification using a
                      Neural Network (between 0 and 1). See the KAL News on WWW.
                      NB! You may want to disable the usual electron ID when
                      using this cut. This can be done with: CUT ELTAG -5
                   0 : default
```

```
CUT ELCON    <num>  : Maximum gamma conversion tag (ELEPHANT) i.e.
                      1 - LOOSE
                      2 - STANDARD
                      3 - TIGHT
                  4 : default

CUT EOP      <num>  : Minimum Energy/Momentum ratio necessary for a
                      particle to be accepted as an electron
                -1.0 : default

CUT PREOP    <num>  : Minimum probability from E/P
                -1.0 : default

CUT DZHPC    <num>  : Maximum of Z(extrapolation) - Z(HPC)
               1000.0 : default

CUT PRSHFIT  <num>  : Minimum probability for shower fit
                -1.0 : default

CUT PRDZTPHP <num>  : Minimum probability for Z(TPC) - Z(HPC)
                -1.0 : default

CUT PRDPHI   <num>  : Minimum probability for phi direction mismatch
                -1.0 : default

CUT PRTPCEL  <num>  : Minimum probability from dE/dX for electron
                -1.0 : default

CUT PRTPCPI1 <num>  : Maximum probability from dE/dX for pion
                -1.0 : default

CUT PRELHPC  <num>  : Minimum electron probability from HPC
                -1.0 : default

CUT PRTPHPEL <num>  : Minimum combined dE/dX and HPC electron prob.
                -1.0 : default

CUT NASHTEL  <num>  : Minimum number of VD hits
                  0 : default

CUT SIGEOVE  <num>  : Maximum Sigma E / E
               1000.0 : default


HPC crack flags:

CUT HPCFLGUT <num>  : -1 - Not accepting tracks outside the HPC range
```

```
                      -2 - No action (default)
CUT HPCFLGPH <num>  :  1 - Phi crack not accepted
                      -2 - No action (default)
CUT HPCFLGTH <num>  :  2 - Theta crack not accepted
                      -2 - No action (default)
CUT HPCFLGPT <num>  :  3 - Phi and Theta crack not accepted
                      -2 - No action (default)

  For vertex fits:
  CUT ELEST &lt;num&gt;  : 0 - Use ordinary (TRAC(8)) track parameters for
                     vertex fits.
                     1 - Use electron refitted track parameters (ELTR(28)) for
                         electron candidates.
                 0 : default


Muon Identification

The following cuts are available for muon identification:

CUT MUTAG   <num>  : Defines the minimum tag necessary i.e.
                     2 - VERY LOOSE
                     3 - LOOSE
                     4 - STANDARD
                     5 - TIGHT
                 3 : default
                 ex: CUT MUTAG 4 means particles tagged as standard
                     or tight are accepted as muons

CUT NASHTMU <num>  : Minimum number of VD hits
                 0 : default

CUT SIGEOVE <num>  : Maximum Sigma E / E
             1000.0 : default



Pion Identification

The following cuts are available for pion identification:

CUT PISIGRIC   <num>  : Defines the minimum RICH tag (HADSIGN) necessary.
                    1 : default
                    ex: CUT PISIGRIC 1 means particles tagged as
                        pions are accepted

CUT PINOINFO   <num>  : Defines the minimum RICH tag (HADSIGN) necessary,
```

```
                        but also  accepts -1 i.e. No Information.
                    -2 : default
                     ex: CUT PINOINFO 1 means RICH tag -1 and 1 are
                        accepted (but not 0)

CUT PRTPCPI     <num>  : Minimum probabiity from dE/dX for pion
                    -1 : default

CUT NASHTPI     <num>  : Minimum number of VD hits
                     0 : default

CUT PIELVETO    <num>  : Maximum eltag (ELEPHANT) (i.e. veto)
                     6 : default

CUT PIMUVETO    <num>  : Maximum mutag (i.e. veto)
                     6 : default

CUT PIKVETO     <num>  : Maximum RICH kaon tag (HADSIGN) (i.e. veto)
                     6 : default

CUT PIPRVETO    <num>  : Maximum RICH proton tag (HADSIGN) (i.e. veto)
                     6 : default

CUT SIGEOVE     <num>  : Maximum Sigma E / E
                1000.0 : default

CUT PITAG1      <num>  : Defines the minimum RICH tag (KHAIDN(1,i)) necessary.
                    -1 : default

CUT PITAG2      <num>  : Defines the minimum RICH tag (KHAIDR(1,i)) necessary.
                    -1 : default

CUT PITAG3      <num>  : Defines the minimum dE/dX tag (KHAIDE(1,i)) necessary.
                    -1 : default

CUT PITAG4      <num>  : Defines the minimum combined tag (RICH - dE/dX)
                         (KHAIDC(1,i)) necessary.
                    -1 : default



Kaon Identification

The following cuts are available for kaon identification:

CUT KASIGRIC    <num>  : Defines the minimum RICH tag (HADSIGN) necessary:
                         1 - LOOSE
```

```
                    2 - STANDARD
                    3 - TIGHT
                1 : default
                ex: CUT KASIGRIC 1 means particles tagged as
                    loose,standard or tight are accepted as kaons

CUT KNOINFO   <num>  : Defines the minimum RICH tag (HADSIGN) necessary, but
                    also accepts -1 i.e. No Information.
               -2 : default
                ex: CUT KNOINFO 1 means RICH tag -1 and 1,2,3 are
                    accepted (but not 0)

CUT KASIGDEX  <num>  : Defines the minimum Kaon signature from dE/dX:
                    1 - LOOSE
                    2 - STANDARD
                    3 - TIGHT
               -2 : default

CUT COMKATAG  <num>  : Defines the minimum combined tag (dE/dX and RICH) (HADSIGN)
               -2 : default

CUT NASHTK    <num>  : Minimum number of VD hits
                0 : default

CUT KELVETO   <num>  : Maximum eltag (ELEPHANT) (i.e. veto)
                6 : default

CUT KMUVETO   <num>  : Maximum mutag (i.e. veto)
                6 : default

CUT KPIVETO   <num>  : Maximum RICH pion tag (HADSIGN) (i.e. veto)
                6 : default

CUT KPROVETO  <num>  : Maximum RICH proton tag (HADSIGN) (i.e. veto)
                6 : default

CUT SIGEOVE   <num>  : Maximum Sigma E / E
             1000.0 : default

CUT KTAG1     <num>  : Defines the minimum RICH tag (KHAIDN(1,i)) necessary.
               -1 : default

CUT KTAG2     <num>  : Defines the minimum RICH tag (KHAIDR(1,i)) necessary.
               -1 : default

CUT KTAG3     <num>  : Defines the minimum dE/dX tag (KHAIDE(1,i)) necessary.
               -1 : default
```

```
CUT KTAG4       <num>  : Defines the minimum combined tag (RICH - dE/dX)
                         (KHAIDC(1,i)) necessary.
                    -1 : default
```

Proton Identification

The following cuts are available for proton identification:

```
CUT PRSIGRIC    <num>  : Defines the minimum RICH tag (HADSIGN) necessary i.e.
                         1 - LOOSE
                         2 - STANDARD
                         3 - TIGHT
                     1 : default
                     ex: CUT PRSIGRIC 1 means particles tagged as
                         loose, standard or tight are accepted as protons

CUT PRNOINFO    <num>  : Defines the minimum RICH tag (HADSIGN) necessary,
                         but also accepts -1 i.e. No Information.
                    -2 : default
                     ex: CUT PRNOINFO 1 means RICH tag -1 and 1,2,3
                         are accepted (but not 0)

CUT PRSIGDEX    <num>  : Defines the minimum Proton signature from dE/dX
                         1 - LOOSE
                         2 - STANDARD
                         3 - TIGHT
                    -2 : default

CUT COMPRTAG    <num>  : Defines the minimum combined tag (dEdX and RICH)
                    -2 : default

CUT NASHTPR     <num>  : Minimum number of VD hits
                     0 : default

CUT PRELVETO    <num>  : Maximum eltag (ELEPHANT) (i.e. veto)
                     6 : default

CUT PRMUVETO    <num>  : Maximum mutag (i.e. veto)
                     6 : default

CUT PROKVETO    <num>  : Maximum RICH kaon tag (i.e. veto)
                     6 : default

CUT SIGEOVE     <num>  : Maximum Sigma E / E
                1000.0 : default
```

```
CUT PRTAG1     <num>  : Defines the minimum RICH tag (KHAIDN(1,i)) necessary.
                   -1 : default

CUT PRTAG2     <num>  : Defines the minimum RICH tag (KHAIDR(1,i)) necessary.
                   -1 : default

CUT PRTAG3     <num>  : Defines the minimum dE/dX tag (KHAIDE(1,i)) necessary.
                   -1 : default

CUT PRTAG4     <num>  : Defines the minimum combined tag (RICH - dE/dX)
                        (KHAIDC(1,i)) necessary.
                   -1 : default
```

Acceptance Cuts for PI0's from kinematic fit:

A Pi0 kinematic fitter is implemented in KAL, fitting any two photons (fulfilling the CUTs) with the Pi0 mass as a constraint. Active use of the CUTs below are recommended, especially CUT DMPI0BF x.xxx which instructs KAL to only fit pairs of photons whose invariant mass before the fit is x.xxx GeV/c^2 from the nominal Pi0 mass.

```
CUT EPHPIMIN <num> : Minimum energy (in GeV) for daughter photon
              0.03 : default

CUT EPHPIMAX <num> : Maximum energy (in GeV) for daughter photon
            1000.0 : default

CUT PI0GAMTH <num> : Maximum ABS(COS(theta) for daughter photon
              1.0  : default

CUT PHOHPC   <num> : Minimum HPC/EMF energy (in GeV)
              0.0  : default

CUT MERGEPHO <num> : Consider neutral showers tagged as Pi0s ('merged showers')
                     by Elephant as photons (1) or Pi0s (0)
              0.0  : default

CUT PI0HEM   <num> : Only use photons in the same hemisphere
                     or both hemispheres for the fit
                     0 - Same hemisphere
                     1 - Both hemispheres
              0    : default
```

```
CUT CHI2PI0  <num> : Maximum chi2 of Pi0 fit
          1.0E32  : default

CUT PI0PROB  <num> : Minimum chi2 probability of Pi0 fit
             0.0  : default

CUT COSPHPI0 <num> : Maximum ABS(COS(theta) for daughter photon
             1.0  : default

CUT PI0PMIN  <num> : Minimum Pi0 momentum (in GeV/c)
             0.0  : default

CUT PI0PMAX  <num> : Maximum Pi0 momentum (in GeV/c)
         10000.0  : default

CUT COS0API0 <num> : Cos (maximum opening angle of two photons
                     before kinematic fit)
            1.0  : default

CUT DMPI0BF  <num> : Maximum mass difference (M(Pi0) - 0.135 GeV/c2) before fit.
           1000.0  : default

CUT FITPI0PH <num> : Refit photons (1) or not (0)
             1.0  : default



Acceptance Cuts for PI0's from common block PSCPI0:

CUT PSC0MIN  <num> : Minimum Pi0 momentum (in GeV/c)
             0.0  : default

CUT PSCMAX   <num> : Maximum Pi0 momentum (in GeV/c)
         10000.0  : default

CUT DMASSPI0 <num> : Maximum mass difference (in GeV/c^2) from
                     nominal Pi0 mass
             2.0  : default



Acceptance Cuts for photons:

CUT SHCOSTH  <num> : Maximum ABS(COS(theta) for photon
             1.0  : default

CUT SHPMIN   <num> : Minimum photon momentum (in GeV/c)
```

```
              0.0  : default

CUT SHPPMAX  <num> : Maximum photon momentum (in GeV/c)
          10000.0  : default

CUT PHOHPC   <num> : Minimum HPC energy (in GeV/c)
              0.0  : default

CUT PHOHAD   <num> : Maximum energy in HCAL (in GeV/c)
          10000.0  : default

CUT MERGEPHO <num> : Consider neutral showers tagged as Pi0s ('merged showers')
                     by Elephant as photons (1) or Pi0s (0)
              0.0  : default


Acceptance Cuts for K0s and Lambdas

CUT XYV0CHI2 <num> : Minimum squared XYZ flight distance divided by
                     it's error squared.
               0.0 : default

CUT PRVTXV0  <num> : Minimum chi2 probability of vertex fit
               0.0 : default

CUT DMASSK0  <num> : Maximum mass difference (in GeV/c^2) from K0s mass
              10.0 : default

CUT DMASSLAM <num> : Maximum mass difference (in GeV/C^2) from Lambda mass
              10.0 : default

CUT V0PMIN   <num> : Minimum V0 momentum (GeV/c)
               0.0 : default

CUT V0PMAX   <num> : Maximum V0 momentum (GeV/c)
            1000.0 : default

CUT V0VETO   <num> : Use (0) or not use (1) charged tracks tagged as
                     V0 daughters
               0.0 : default

NB! CUTs have to be defined before IDENTV0.
```

Visible energy

The variables **evish1, evish2, m2h1** and **m2h2** contain respectively the visible

energy and invariant mass squared in hemisphere 1 and 2 (wrt the b-tag thrust axis). The default is that all charged tracks are pions, and all electromagnetic showers are photons. Hadronic showers are not included. By using: CUT EVISHAD 1 hadronic showers will also be included. With CUT EVISPAID 1 particle identification will be used to assign a mass hypothesis (muon, kaon, pion, proton, electron) to each particle. In this case tracks tagged as V0 daughters will not be used, instead the V0 hypothesis is retained. NB! This routine uses all the tracks in the VECP array i.e. it will be affected by the TRACK selection flag TRKSEL in SKELANA.

```
Flags to control program execution

CUT NTHEVENT <num> : ‘‘Modulo’’ for Kal variable NTHEVENT
           10000   : default
                     can be used to perform an action every
                     <num> events
                     Example: print run # every 100 events

                         cut nthevent 1000
                         if nevents = 0 show run



Flags to control output / error messages

CUT VTXOUT  1       : Each time a vertex fit fails it
                      will print a warning message
                      giving the exact error number.
            n > 1   : Will print the warning message only
                      in the n first events
           -1       : No output (default)

CUT ERRMAOUT  1       : KAL inverts the weight matrix for
                        each charged track to get the error matrix.
                        Each time this matrix inversion fails
                        it will print a warning message.
              n > 1   : Will print the warning message only in
                        the n first events
             -1       : No output (default)

CUT PIODEBUG  1       : Prints out information from the Pi0 fit.
              n > 1   : Output only in the n first events
             -1       : No output (default)



KAL DIRECTIVE: Beginning of a script
```

```
The script MUST begin with: KAL ON


KAL DIRECTIVE: Call the DELPHI Inclusive B Physics Program Bsaurus [9]

The user may call the DELPHI Inclusive B Physics Program Bsaurus with the
directive BSAURUS ON in the beginning of the script (right after KAL ON).

;Beginning of a KAL script:

kal on

bsaurus on
```

C KAL Variables

- Comments on KAL variables
- Predefined variables
- User variables
- Variable names
- Variables which are defined everywhere in the program (with Bsaurus variables)
- Variable which is defined after JBIT / JBYT
- Variables defined after SELECT / SELCC
- Kinematic variables of the whole selected system
- Kinematic variables of daughter particles
- Variables defined after SAVE/SAVEFITM
- Variables defined after FITE
- Variables defined after FITALL and FITVTX
- Variables defined after PRIMVTX
- Variables defined for measured particles only
- Variables defined for neutral decaying particles
- Variables defined for PI0's from kinematic fit
- Variables defined for Monte Carlo data

Comment:

Due to the structure of KAL, arrays such as VECP (...) are not necessary. The corresponding quantity is available as variable **vecp...** within a SELECT loop. In fact, arrays are not known to the KAL system. For all measured tracks, nearly all the variables in the DELPHI common blocks are available. The name of the variable is the array name plus the last three letters in the corresponding pointer. For example will the x-component of the momentum of track number **i**, which is placed in **vecp (1,i) = vecp**

(iparpxx,i), be called **vecppxx** in KAL. The same way, the impact parameter in $R\Phi$ will be **qtracimp.** Note that variable names that are not arrays have the same name in KAL as in the standard commons i.e. **nevent, nvecp ...** are called **nevent, nvecp ...** In the case where there are no pointer (like for **vecp (2,i)** , the usual convention has been 'extrapolated' i.e. the y-component of the momentum is called **vecppyy** . Appendix C contains a complete list of all the variables available in KAL. A complete list of what the SKELANA variables are called in KAL are available from the KAL main www page: Variable names.

There exist about 800 PREDEFINED variables which are under control of the KAL system. Their value depend on the context of statements :

- Some variables are defined for the whole event i.e they can be used everywhere in the program, e.g. **nevent** (event number).
- Most of the variables are defined only by a preceding SELECT statement, e.g. all variables which describe the kinematics of the selected system (**mass, e, p, px, py** ...).
- PREDEFINED variables must not be redefined. However, some PREDEFINED variables can be changed via the SET statement.

SET statement:

There are a few PREDEFINED variables which are explicitely allowed to be redefined by a SET statement :

```
SET ECMAS = <expr>

SET FLAG     = <expr>
SET USERVA01 = <expr>
....
SET USERVA10 = <expr>
SET WEIGHT   = <expr>

SET P    = <EXPR>
SET PX   = <EXPR>
SET PY   = <EXPR>
SET PZ   = <EXPR>
SET E    = <EXPR>
SET MASS = <EXPR>

SET P = 1    The momentum vector of the selected system will be
             a unit vector, i.e. Px**2 + Py**2 +Pz**2 = 1
SET P = - P  The momentum is rotated by 180 degrees, i.e. :
             Px -> -Px, Py-> -Py, Pz -> -Pz. The variable P
             remains unchanged (positive) !
```

All other PREDEFINED variables should not be redefined in the program, i.e. they should not appear on the left side of an assignment sign '=' nor in a directive command.

If they do, however, they are assumed to be assigned variables which are not under control of the KAL system any more. A warning message will be printed as soon as a PREDEFINED variable is redefined in the program.

User variables:

User (assigned) variables can be used in a similar way as FORTRAN variables : their value is defined by assignment statements or by a DATA directive, e.g. :

```
BASURA = SQRT ( 1 - COSTHETA ** 2 )
DATA MUELL 0.5
```

Note that the DATA directive defines and initializes at 'compile' time. During the program execution, MUELL might be modified by your KAL program. MUELL won't be reinitialized at the beginning of the processing of the following event!

An assigned variable can be used in an arithmetic expression (e.g. on the right side of an '=' sign) only if it has been established before (i.e. if it appears on the left side of a '=' sign or in a DATA directive before).

Names of variables may consist of up to eight characters. All variables are floating point variables.

Variable names:

KAL User variables may be up to eight characters long. The name has to be different from the name of PREDEFINED variables. Some other restrictions are:

( ) and , are not allowed anywhere in the variable name.
Numbers and the characters * + - / are not allowed in first position.

Variables which are defined everywhere in the program:

- Variables from PSCEVT
- Variables from PSCUTT
- Variables from PSCFLG
- Variables from PSCBTG
- Variables from PSCLUM
- Variables like **nvtx, nvecp , nhpc** etc ie the number of tracks/entries in the different arrays in stdcdes.car/SKELANA

```
Numericals

#PI         pi (3.14...)
#180/pi     180 / pi
```

```
#E          e (2.7...)
#INF        infinity = 1.E32
#ZERO       0


Run specific variables

IIIEXP      Exp. number
IIIRUN      Run number
IIIEVT      Event number
IIFILE      File
IIIDAT      Date
IIITIM      Time
IIFILL      Fill number
ECMAS       Centre-of-mass energy
NEVENT      Number of events read from the input file
NTHEVENT    Mod(NEVENTS,number defined by CUT NTHEVENT)
NBEVTOUT    Total number of events written to output stream (stripping)


Visible hemisphere energy:

EVISH1      Visible energy in hemisphere 1 (wrt b-tag thrust axis)
EVISH2                          .....     2
M2H1        Invariant mass squared in hemisphere 1 (wrt b-tag thrust axis)
M2H2                            .....             2
            (see also CUT EVISHAD and CUT EVISHAID)

ENEMH1      Electromagnetic (HPC & EMF) calorimeter energy in hemisphere 1
ENEMH2          ‘‘      ‘‘          ‘‘            ‘‘      ‘‘          ‘‘    2
ENHACH1     Hadronic calorimeter energy in hemisphere 1
ENHACH2         ‘‘      ‘‘          ‘‘            ‘‘    2


Btag information:

BCONFEV     Event b-confidence (from Bill Murray b-confidence)
            NB! The normal btagging should have run before in SKELANA
                (ie put IFLBTG = 2 and IFLBSP = 2)


Bsaurus variables (NB! You need the directive BSAURUS ON to fill these variables):

ISAURUS1  Hemisphere status words from BSAURUS
ISAURUS2  (Hemisphere 1 is defined by the highest energy jet)

BSEBTAG   Event b-tag
```

```
BSEBCONF  Event b-confidence (Bill Murray)
BSEBCTAG  Event b-tag with charm suppression
BSENJET   Number of jets with B-jet algorithm
BSEFLAV   MC primary quark flavour (e.g. 12 = b)

BSEPVX    Primary vertex X
BSEPVY                   Y
BSEPVZ                   Z

BSEPC1    Primary
  -          vertex
BSEPC6         covariance matrix

BSEMCPVX  MC primary vertex X
BSEMCPVY                    Y
BSEMCPVZ                    Z


Hemisphere information: n = 1,2 for hemisphere 1,2

BSHnBTAG  Hemisphere b-tag
BSHnBCON  Hemisphere b-confidence (Bill Murray)
BSHnBCTG  Hemisphere b-tag with charm suppression

BSHnBF1   B 4-vector
  -         from
BSHnBF4       fit

BSHnBT1   B 4-vector
  -         from
BSHnBT4       fit + energy corrections

BSHnBY1   B 4-vector
  -         from
BSHnBY4       rapidity algorithm

BSHnSVX   Secondary vertex: X
BSHnSVY                     Y
BSHnSVZ                     Z

BSHnSC1   Secondary
  -         vertex
BSHnSC6       covariance matrix

BSHnDEC   B decay length
BSHnDECS  B decay length error
```

```
BSHnPROB  Secondary vertex fit probability
BSHnRET   Secondary vertex fit return code (error flag)
BSHnNACC  Number of particles at secondary vertex
BSHnQJET  Jet charge
BSHnQVER  B charge neural network
BSHnQVES  B charge error
BSHnFLAV  b flavour neural network: +1 = good b
                                    -1 = good bbar
BSHnMY    Rapidity mass
BSHnSHEM  Scaled measured hemisphere energy
BSHnQUAL  Hemisphere quality flag
BSHnBDEC  MC generated 3-d B decay lenght
BSHnDDEC  MC generated 3-d Dbar decay lenght
BSHnDPDE  MC generated 3-d D decay lenght

BSHnBVX   MC generated B vertex: X
BSHnBVY                           Y
BSHnBVZ                           Z

BSHnDVX   MC generated Dbar vertex: X
BSHnDVY                             Y
BSHnDVZ                             Z

BSHnDPVX  MC generated Dvertex: X
BSHnDPVY                        Y
BSHnDPVZ                        Z

BSHnMVER  Mass on secondary vertex
BSHnBPLU  B+ tagging



Variable defined after JBIT / JBYT

The KAL statements JBIT and JBYT return certain bits or consecutive
number of bits. Since FORTRAN-like statements like

      yesno = jbit(var,ibit)

are unkown to kal, the result of the JBIT/JBYTE statements is stored in
the KAL-variable RESULT.


  RESULT     contains output of JBIT/JBYTE operations
```

```
Variables defined after SELECT / SELCC

(1) Kinematic variables of the selected system

CHARGE     Charge

P          Momentum
PX         Px
PY         Py
PZ         Pz
PT         Transverse momentum wrt beam

E          Energy
MASS       Mass of the whole system

PHI        Phi = azimuth
COSTHETA   Cos(theta) = cos(polar angle)

CHI2       Chi**2 = (meas mass - mass(part.table))**2/ error**2
           (which is only defined if the particle has been SAVEd before!)
           After a FITE statement, it contains:
           (meas energy - energy to be fitted)**2 / (error on E)**2.
           The error is in both cases the measured error.

NDAUGHT    Number of daughter particles (0 ... 4)

CC      +1 If the charged conjugated system is selected (after SELCC)
        -1 If not

FLAG       Track flag free to be used for any purpose. Default = 0.
           May be redefined by the user)
USERVA01   As
...         for
USERVA10      FLAG
WEIGHT     Weight. Default value = 1. Its value may be redefined.
           Weight (composite particle) = PROD (weight (daughters))

TRACKNOK   Particle number in /CK/ buffer
TRACKNOM   Particle number in /CM/ buffer
TRACKNO    Particle number in stdcdes arrays (like VECP etc)

PXPHOTCO   PXPHOT code of track

BTAGTRK    Borissov probability for (charged) track - Single track
           uds probability, with the following meaning:
           Near 0 : large positive lifetime, probably from B track
```

```
                1 : fits perfectly on main vertex (or no good information)
             > 1-2 : negative lifetime (unphysical, can only be resolution)
            Enrichment of b-tracks ONLY by using very small BTAG values
            (<0.05 or so)
BCONF       Bill Murray's b-confidence for (charged) track: Likelihood
            ratio of the track coming from a b event compared to all other
            hypotheses (uds,c). b-enrichment above the unbiased value 0.2
NB! For the two last variables the normal btagging should have run before
    in SKELANA (i.e. put IFLBTG = 2 and IFLBSP = 2))


Variables defined after SELECT / SELCC

(2) Kinematic variables of daughter particles

NB! If you use refitted photons (i.e. CUT FITPI0PH 1.0 ) to make Pi0s
this information may be incorrect since it involves the original momenta
of the photons i.e. M12 and P12 will give the ``Pi0'' mass and momentum
based on the original (unfitted) photon momenta etc. Be carefull about NOT
using LV01,LV02,V01,V02 since this will mix the fitted Pi0 momentum with the
original photon momenta. For these angles use PI0LV01, PI0LV02,PI0V01,PI0V02.
Also note that when selecting the Pi0 you will get the fitted Pi0 mass and
momentum which will be used in all particles where the Pi0 is included.
Then selecting the photon daughters will give you the original photon momenta.


P1          momentum of the 1st daughter particle
P2          momentum of the 2nd daughter particle
P3          momentum of the 3rd daughter particle
P4          momentum of the 4th daughter particle

            The numbers 1 ...  4  refer to the order in which the
            particles are defined in the SELECT / SELCC statement.

            This number should not exceed the number of selected
            particles Example :
                            SELECT K+ K-
                            F0 = P  ;  = momentum of the K+ K- system
                            F1 = P1 ;  = K+ momentum
                            F2 = P2 ;  = K- momentum
                            F3 = P3 ;  NOT defined !!!!!!!
                            F4 = P4 ;  NOT defined !!!!!!!

E1          energy of the 1st daughter particle
E2          energy of the 2nd daughter particle
E3          energy of the 3rd daughter particle
E4          energy of the 4th daughter particle
```

```
M1          mass of the 1st daughter particle
M2          mass of the 2nd daughter particle
M3          mass of the 3rd daughter particle
M4          mass of the 4th daughter particle

P12         momentum of the subsystem consisting of
            the 1st and 2nd daughter particles
P13                       ...
P14                       ...
P23                       ...
P24                       ...
P34                       ...

E12         energy of the subsystem consisting
            the 1st and 2nd daughter particles
E13                       ...
E14                       ...
E23                       ...
E24                       ...
E34                       ...

M12         invariant mass of the subsystem consisting of
            the 1st and 2nd daughter particles
M13                       ...
M14                       ...
M23                       ...
M24                       ...
M34                       ...

MM12        (invariant mass)**2 of the subsystem consisting of the
            1st and 2nd daughter particles (Dalitz plots !)
MM13                      ...
MM14                      ...
MM23                      ...
MM24                      ...
MM34                      ...

P123        momentum of the subsystem consisting of
            the 1st, 2nd, and 3rd daughter particles
P124                      ...
P134                      ...
P234                      ...

E123        energy of the subsystem consisting of
            the 1st, 2nd, and 3rd daughter particles
E124                      ...
```

```
E134                    ...
E234                    ...

M123        invariant mass of the subsystem consisting of
            the 1st, 2nd, and 3rd daughter particles
M124                    ...
M134                    ...
M234                    ...

V01         cos(Angle) between the whole system and the 1st daughter
            particle (lab system)
V02         cos(Angle) between the whole system and the 2nd daughter
V03         cos(Angle) between the whole system and the 3rd daughter
V04         cos(Angle) between the whole system and the 4th daughter

V12         cos(Angle) between 1st and 2nd daugther (lab system)
V13         cos(Angle) between 1st and 3rd daugther
V14         cos(Angle) between 1st and 4th daugther
V23         cos(Angle) between 2nd and 3rd daugther
V24         cos(Angle) between 2nd and 4th daugther
V34         cos(Angle) between 3rd and 4th daugther

LV01        cos(Angle) between the whole system and the 1st daugther
            (after a Lorentz boost into the rest frame of the whole
             system)
LV02        cos(Angle) between the whole system and the 2nd daugther
LV03        cos(Angle) between the whole system and the 3rd daugther
LV04        cos(Angle) between the whole system and the 4th daugther

LV1D2       Assume a decay of A --> B C and a subsequent decay B -> E F
            LV1D2 is the cosine between C and E in the rest frame of B
LV2D1       Assume a decay of A --> B C and a subsequent decay C -> E F
            LV1D2 is the cosine between B and E in the rest frame of C



Variables defined after SAVE - SAVEFITM

ACCEPT      = +1 : particle saved
            = -1 : not saved (due to cuts DMASS)

DMASS       mass difference = meas. mass - mass(part. table)

ERROR       = -1 : Fit ok
            > 1  : Fit ended erroneously
             = +2 : Inconsistent derivative vector
```

```
             = +3 : Fit diverged
             = +4 : Max. number of iterations
             = +5 : Neg. diagonal element in error matrix
              (only defined after SAVEFITM, not after SAVE !)

CHI2         chi**2 = (meas mass - mass(part. table))**2 / error**2
             (CHI2 is stored, in contrast to DMASS and ERROR, and can
             be used in subsequent loops). The error is the measured error.



Variables defined after FITE:

ERROR        = -1 : fit ok
             = +1 : fit ended erroneously

CHI2         Delta(E)**2 / error**2
             (CHI2 is stored, in contrast to DMASS and ERROR,
             and can be used in subsequent loops). The error
             is the measured error.



Variables defined after FITALL, FITVTX and MAKE2VTX

ERROR        = Actual error from VDF2ND

ACCEPT       = +1 : fit ok
             = -1 : fit ended erroneously

ALLVXX       x position of fitted vertex (cm)
ALLVXY       y position of fitted vertex (cm)
ALLVXZ       z position of fitted vertex (cm)

CHI2FA       Chisquare of fit

PRCHI2FA     Chisquare probability of fit
             (from the CERNLIB routine PROB (CHI2,NDOF),
             NDOF = (NDIM - 1) * L - NDIM
             where: NDOF = Number of degrees of freedom
                    NDIM =   ``    `` dimensions of fit
                    L    =   ``    `` tracks used in fit

TESTCHI2     Contribution to the total chisquare from the worst track
TESTTRKM     Track number (CM) of worst track
RESULT       Number of tracks (temporary variable!)
```

The following variables are only defined for FITALL and MAKE2VTX:

VTX2CO1 .... VTX2CO6	Covariance matrix on x-, y- and z-position of the fitted vertex
THETANEW	New theta
PHINEW	New phi
KAPPANEW	New curvature
PARERR1 .... PARERR6	Measured error matrix
VPXFIT	New Px
VPYFIT	New Py
VPZFIT	New Pz
VPFIT	New P

Variables defined for MEASURED PARTICLES only

The following variables are defined:

(1) in a SELMEAS loop
(2) in any SELECT loop which selects exactly one measured particle (e.g. SELECT PI+).

Some variables are defined for decaying particles with a reconstructed secondary vertex (see below).

Variables from PSCVEC
Variables from PSCTRA
Variables from PSCHPC
Variables from PSCEMF
Variables from PSCHAC
Variables from PSCCAL
Variables from PSCMUD
Variables from PSCMUV
Variables from PSCMUL
Variables from PSCMUT
Variables from PSCELD
Variables from PSCELO
Variables from PSCHAD

```
Variables from PSCDEX
Variables from PSCGRC
Variables from PSCLRC
Variables from PSCELT
Variables from PSCVDA (only NASSHIT (= NASHT))

DR          Radial distance main vertex - track
DZ          Delta z main vertex - track

KAPPA       Curvature of the track = 1/radius
D0          Impact parameter RPhi
Z0          Impact parameter Z
THETA       Theta at d0
PHI0        Phi   at d0


IVECPHEM    Hemisphere of particle wrt sphericity axis
IVECPJET    Jet number
PTJET       Transverse momentum of particle wrt jet
            (having removed the particle from the jet)

JET1X       X-component of jet number 1
JET1Y       Y-component of jet number 1
JET1Z       Z-component of jet number 1
 ...
JET7X       X-component of jet number 7
JET7Y       Y-component of jet number 7
JET7Z       Z-component of jet number 7

SPHERX      X-component of sphericity axis
SPHERY      Y-component of sphericity axis
SPHERZ      Z-component of sphericity axis

THRUSTX     X-component of thrust axis
THRUSTY     Y-component of thrust axis
THRUSTZ     Z-component of thrust axis



C.J. Kreuter's new Electron Identification using a
Neural Network is available as: ELECNID



NEW (HADRON) TAGS

  The new tags (XNEWTAG,RPRORI,RPRODE and RPROCO) are now available in SKELANA
```

(in the common block PSCHAD). The name of the variables in KAL is the name of the array followed by the entry number i.e. the SKELANA variable KHAIDN(1,i) is called 'khaidn1' in KAL. These new names replace the variables previously called IHANEn, JHANEn, IHADEn, JHADEn, IHACOn and JHACOn (but the content is of course the same). This change is due to the fact that these variables were available in KAL a long time before they became available in SKELANA. Now that they are the KAL names have been changed to be consistent with the usual name convention of SKELANA variables.

Variables defined for PI0s from kinematic fit:

NCCP0	Number of pi0 candidate
OAP0	Pi0 opening angle
CHSQ0	Chi2 of kin. fit
PI0PROB	Probability of fit
JDAUPI01	Index (Vecp) of gamma daughter 1
JDAUPI02	Index (Vecp) of gamma daughter 2
MPI0BF	Invariant mass of the two gammas before kin. fit
PI0V01	Cos(Angle) between the Pi0 and the 1st (refitted) daughter photon (lab system)
PI0V02	Cos(Angle) between the Pi0 and the 2nd (refitted) daughter photon (lab system)
PI0LV01	Cos(Angle) between the Pi0 and the 1st daugther photon (after a Lorentz boost into the rest frame of Pi0).
PI0LV02	Cos(Angle) between the Pi0 and the 2nd daugther photon (after a Lorentz boost into the rest frame of Pi0).
PI0PULL1	(E(gamma 1, new) - E(gamma 1, old)) / error
PI0PULL2	(Theta(gamma 1, new) - Theta(gamma 1, old)) / error
PI0PULL3	(Phi(gamma 1, new) - Phi(gamma 1, old)) / error
PI0PULL4	(E(gamma 2, new) - E(gamma 2, old)) / error
PI0PULL5	(Theta(gamma 2, new) - Theta(gamma 2, old)) / error
PI0PULL6	(Phi(gamma 2, new) - Phi(gamma 2, old)) / error

The error is the measured error.

PXP0D1	Refitted Px for gamma 1
PYP0D1	Refitted Py for gamma 1
PZP0D1	Refitted Pz for gamma 1
PXP0D2	Refitted Px for gamma 2
PYP0D2	Refitted Py for gamma 2
PZP0D2	Refitted Pz for gamma 2

For PI0's from PSCPI0, all variables in the common block is defined.

Variables defined for neutral decaying particles (V0s)

Variables from PSCRV0 and:

KRV0HEM1 : Hemisphere of daughter particle 1
KRV0HEM2 : Hemisphere of daughter particle 2

Variables defined for the PRIMARY VERTEX:

NPVITER Number of iterations used in PNPVFIT

PVIERR Error flag: 0 means fit OK
(see DSTANA writeup for more details)

PVNTK Number of tracks used in fit
PVNDF Number of degrees of freedom of fit

PVX X-coordinate of primary vertex (cm)
PVY Y-coordinate of primary vertex (cm)
PVZ Z-coordinate of primary vertex (cm)

PVCH2 Chi2 of vertex fit

PVERR1 Measured
.... error
PVERR6 matrix

See the DSTANA writeup for information about PNPVFIT.

Variables defined for Monte Carlo data

In the FRAME MC: - All usual kinematic variables (P, M, ...)
- MCX Starting
MCY point of
MCZ track
- MCVECLIN Vecp track index number (0 if none)

The following variables are defined for ‘‘measured’’ particles i.e.

not in the ‘‘frame mc’’: (if there is no link from a measured track
to a generated one, all these variables will have the value 0)

KP1	Status code of current particle
KP2	Particle code (Jetset convention)
KP3	Index of mother particle
KP4	Index of first daughter particle
KP5	Index of last daughter particle
PP1	Px in GeV
PP2	Py in GeV
PP3	Pz in GeV
PP4	Energy in GeV
PP5	Mass in GeV
VP1	X-position of production vertex in mm
VP2	Y-position of production vertex in mm
VP3	Z-position of production vertex in mm
VP4	Time of production
VP5	Proper lifetime of the particle
KP2MOTH	Particle code of mother
INDMOTH	Track index in VECP array of mother
PP1MOTH	Px of mother
PP2MOTH	Py of mother
PP3MOTH	Pz of mother
PP5MOTH	Mass of mother
KP2GRAMO	Particle code of grandmother
INDGRAMO	Track index in VECP array of grandmother
KP2GGRMO	Particle code of grandmother’s mother
INDGGRMO	Track index in VECP array of grandmother’s mother