DST Analysis Libraries Writeup

DELPHI Analysis Software Task Groups

May 15, 1997

Contents

1 Introduction

This is a LaTeX file, normally called

```
VXCERN::DISK$DELPHI3:[DPADEV.WUP]DSTANA.TEX
```

which will be updated as more routines are added to the libraries. If you want to add another routine, copy the files

```
VXCERN::DISK$DELPHI1:[DAUNCEY.TEX]EXAMPLE_ROUTINE.TEX  or
VXCERN::DISK$DELPHI1:[DAUNCEY.TEX]EXAMPLE_FUNCTION.TEX
```

as appropriate and modify it for the new routine. Send an email to `DAUNCEY@V2.RL.AC.UK` or `MIRABITO@LYOHP5.IN2P3.FR` and it will be added to this writeup.
This note contains documentation of Delphi routines for post-DST analysis. The routines are divided up into sections corresponding to related routines. Each routine has one page, with a description of the calling sequence, arguments and usage notes. Each routine has === marks above and below the listed arguments, indicating the argument is input (above) and/or output (below). Any problems, comments or proposed modifications about a routine should be directed to the person(s) whose name appears at the bottom of the routine page.
In principle, only routines which are usable in all offline packages used in Delphi should be reported here. This means they should give the same results called within PXDST, PHDST, PHYANA etc. They must not assume that anything beyond the "standard" commons are already filled. See section 2 for information on these commons.
For details about the structure of the DST data, see the PXDST manual.

2 Standard Commons

At the moment, the only commons which are used throughout the offline packages are the following.

`/UXCEVT/`	Contains run, event, file sequence and fill numbers (amoung others).
`/UXLINK/`	Contains the link to the start of the DST data, `LDTOP`.
`/UXLRUN/`	Contains the links to the start of the SOR/EOR data, `LSTRUN, LEORUN`.
`/UXCOM/`	Contains the Zebra data arrays, `Q(*)` and `IQ(*)`.
`/PXCHDR/`	Contains the pilot record.
`/PXCONS/`	Contains some constants (including π) and the beam energy and magnetic field in Tesla.

In addition, the two commons used by the VECSUB routines are

`/PUCPPP/`	Contains the momenta of the tracks
`/PUCLLL/`	Contains the PA links of the tracks

Also, the VDCLAP library uses the commons

`/VDIOTD/`	Contains the details of the VD hits
`/VDIOTK/`	Contains the details relevant to the VD of the tracks

These commons are described further is this section.

2.1 UXCEVT

Information: Run and event numbers and counters for events in the job.
Source Code: Keep UXCEVT, Patch UXCDES, Cards UX, Disk UTY.
Common Structure:

```
+KEEP , UXCEVT .
        INTEGER*4       NEXP   , NROLD  , IEVOLD , NRUN   , IEVT   ,
     +                  NFILSQ , IEVDAT , IEVTIM , NEVT   , NEVTOT ,
     +                  NCPT   , NFIOLD , NFILL  , NVERDS , IPRDAT ,
     +                  IPRTIM
        COMMON /UXCEVT/ NEXP   , NROLD  , IEVOLD , NRUN   , IEVT   ,
     +                  NFILSQ , IEVDAT , IEVTIM , NEVT   , NEVTOT ,
     +                  NCPT   , NFIOLD , NFILL  , NVERDS , IPRDAT ,
     +                  IPRTIM
```

Important Variables:

INTEGER	NRUN	Run number
INTEGER	IEVT	Event number
INTEGER	NFILSQ	Online tape (file sequence) number
INTEGER	IEVDAT	Event date
INTEGER	IEVTIM	Event time
INTEGER	NEVT	Number of events in this run in the job
INTEGER	NEVTOT	Total number of events in the job
INTEGER	NFILL	Fill number
INTEGER	NVERDS	PX version number $\times 100$
INTEGER	IPRDAT	DELANA processing date
INTEGER	IPRTIM	DELANA processing time

Person Responsible: UX responsible (?)

2.2 UXLINK

Information: Links to different event data structures.
Source Code: Keep `UXLINK`, Patch `UXCDES`, Cards `UX`, Disk `UTY`.
Common Structure:

```
+KEEP , UXLINK .
        INTEGER          LTEMP  ,LRTOP  ,LSTOP  ,LTTOP  ,LITOP  ,LRTEMP ,
     +                   LRWTMP ,LRAWUX ,LBKTOP ,LORTOP ,LRTINT ,LDTOP
        COMMON /UXLINK/ LTEMP(2) ,LRTOP ,LSTOP ,LTTOP ,LITOP ,LRTEMP ,
     +                   LRWTMP ,LRAWUX ,LBKTOP ,LORTOP ,LRTINT ,LDTOP
```

Important Variables:

`INTEGER`	`LRTOP`	Link to top of raw data structure
`INTEGER`	`LSTOP`	Link to top of simulation data structure
`INTEGER`	`LDTOP`	Link to top of DST data structure

Person Responsible: UX responsible (?)

2.3 UXLRUN

Information: Links to Start-Of-Run (SOR) and End-Of-Run (EOR) data structures.
Source Code: Keep `UXLRUN`, Patch `UXCDES`, Cards `UX`, Disk `UTY`.
Common Structure:

```
+KEEP , UXLRUN .
        INTEGER    NCPTMX
        PARAMETER (NCPTMX=100)
        INTEGER          LRUNLK  ,LSTRUN,LEORUN,LCORUN,LCPT,
     +                            LSTTMP,LENTMP,LCOTMP,LCPTMP,
     +                            LORSTR,LOREOR,LORCOR,LORCPT
        COMMON /UXLRUN/ LRUNLK(2),LSTRUN,LEORUN,LCORUN,LCPT(NCPTMX),
     +                            LSTTMP,LENTMP,LCOTMP,LCPTMP(NCPTMX),
     +                            LORSTR,LOREOR,LORCOR,LORCPT(NCPTMX)
```

Important Variables:

`INTEGER`	`LSTRUN`	Link to top of SOR data structure
`INTEGER`	`LEORUN`	Link to top of EOR data structure

Person Responsible: UX responsible (?)

2.4 UXCOM

Information: ZEBRA arrays containing data.
Source Code: Keep UXCOM, Patch UXCDES, Cards UX, Disk UTY.
Common Structure:

```
+KEEP , UXSIZEQ .
*- Dimension of UX store + working space
        INTEGER     NSIZEQ           , NSIZWS
        PARAMETER ( NSIZEQ = 1400000 , NSIZWS = 350000 )

+KEEP , UXQUES .
        INTEGER        IQUEST
        COMMON /QUEST/ IQUEST(100)

+KEEP , UXCOM .
+CDE , UXSIZEQ .
+CDE , UXQUES .
        REAL     Q(NSIZEQ)
        INTEGER IQ(NSIZEQ) , LQ(NSIZEQ)
        REAL     UXFENC
        INTEGER  LUXSTR , LUXREF
        EQUIVALENCE ( LQ(1) , LUXSTR(1) ) , ( Q(1) , IQ(1) , LQ(9) )
        COMMON /UXCOM/ UXFENC(10) , LUXSTR(30) , LUXREF(30)
```

Important Variables:

INTEGER	NSIZEQ	Size of Zebra array
INTEGER	LQ	Link part of Zebra array
INTEGER	IQ	Integer part of Zebra array
REAL	Q	Real part of Zebra array

Person Responsible: UX responsible (?)

2.5 PXCHDR

Information: ZEBRA pilot record.
Source Code: Keep `PXCHDR`, Patch `PXCDES`, Cards `PXDST`, Disk `PHY`.
Common Structure:

```
+KEEP, PXCHDR.
      INTEGER NWPXMA,NWPILT,IUPILT
      PARAMETER(NWPXMA=1024)
      COMMON /PXCHDR/ NWPILT, IUPILT(NWPXMA)
```

Important Variables:

`INTEGER`	`NWPILT`	Number of words in pilot record
`INTEGER`	`IUPILT`	Array containing pilot record

Person Responsible: Y. Sacquin

2.6 PXCONS

Information: Various constants
Source Code: Keep PXCONS, Patch PXCDES, Cards PXDST, Disk PHY.
Common Structure:

```
+KEEP, PXCONS.
* General Constants.
      COMMON /PXCONS/ BMAG, AMPI, PI, EBEAM, EBMDEF
      PARAMETER( EPS=1.0E-20)
      PARAMETER( CONSA = .0029979246 )
```

Important Variables:

REAL	BMAG	Magnetic field in Tesla
REAL	AMPI	Pion mass in GeV/c^2
REAL	PI	π
REAL	EBEAM	Beam energy
REAL	EBMDEF	Default beam energy
REAL	EPS	Very small value
REAL	CONSA	Conversion factor to convert magnetic field from Tesla to GeV/cm

Usage Notes: PXCONS does not have explicit typing for the variables included in the keep sequence.
Person Responsible: Y. Sacquin

2.7 PUCPPP, PUCLLL

Information: Momenta and other information for tracks.
Source Code: Keeps PUCPPP, PUCLLL, Patch VECDES, Cards VECSUB72, Disk PHY.
Common Structure:

```
+KEEP, MTRACK.
      INTEGER MTRACK
      PARAMETER (MTRACK=300)

+KEEP, PUCPPP.
*  Common block used by the VECSUB package
+CDE,MTRACK.
      COMMON /PUCPPP/VECP(10, 2*MTRACK)
      REAL VECP
      INTEGER IVECP(10, 2*MTRACK)
      DOUBLE PRECISION PVECP(5, 2*MTRACK)
      EQUIVALENCE ( VECP(1, 1), PVECP(1, 1))
      EQUIVALENCE (IVECP(1, 1),  VECP(1, 1))

+KEEP, PUCLLL.
*  Common block used to store PA links for the tracks
      COMMON /PUCLLL/LVECP(2*MTRACK)
      INTEGER LVECP
```

Important Variables:

INTEGER	MTRACK	Maximum number of tracks allowed
REAL	VECP(10,*)	Real variables for each track (see below)
INTEGER	IVECP(10,*)	Integer variables for each track (see below)
INTEGER	LVECP(*)	Links for PA bank of each track

Usage Notes: The standard meaning of the elements of the VECP array are as follows.

VECP(1,I)	= x component of momentum of track I
VECP(2,I)	= y component of momentum of track I
VECP(3,I)	= z component of momentum of track I
VECP(4,I)	= energy of track I
VECP(5,I)	= mass of track I
VECP(6,I)	= absolute momentum of track I
VECP(7,I)	= charge of track I
VECP(8,I)	= mass code (delphi convention) of track I
VECP(9,I)	= position in VECP of associated particle
IVECP(10,I)	= jet flag (integer). This is set by the jet-finding routines
	= 1000*(jet number)+100*(hemisphere number)

There are two times MTRACK elements in the arrays. The standard usage is to use the first half for manipulating the reconstructed tracks and the second half for the simulation truth tracks.
Persons Responsible: Y. Sacquin, P. Dauncey

2.8 VDIO, VDIOTD, VDIOTK

Information: VD information for tracks and VD hits.
Source Code: Keeps `VDIO, VDIOTD, VDIOTK` Patch `VDKEEP`, Cards `VDKEEP`, Disk `MVTX`.
Common Structure:

```
+KEEP, VDIO.  **********************************************************
+CDE, VDIOEV.
+CDE, VDIOTD.
+CDE, VDIOTK.
+KEEP, VDIOTD.  ********************************************************
*     Vertex detector hit data.
*
+CDE, VDIOTDP.  Parameters defining elements within the arrays.
+CDE, VDIOTDC.  Definition of the COMMON block.
+KEEP, VDIOTDC. ********************************************************
      INTEGER    MAXTD, MXVTD
      PARAMETER (MAXTD= 1000)          ! Maximum number of hits
      PARAMETER (MXVTD=   80)          ! Storage allocated per hit
*
      INTEGER NVDTD,NVDTDX,NVDTDZ      ! Number of hits this event
      REAL     VDTD(MXVTD,MAXTD)       ! Floating-point TD data
      INTEGER IVDTD(MXVTD,MAXTD)       ! Integer TD data
*
      EQUIVALENCE  (IVDTD, VDTD)       ! Both stored in the same array
*
      COMMON /VDIOTD/ NVDTD,NVDTDX,NVDTDZ,VDTD
+KEEP, VDIOTK.  ********************************************************
*     Tracks through the vertex detector.
*
+CDE, VDIOTKP.  Parameters defining elements within the arrays.
+CDE, VDIOTKC.  Definition of the COMMON block.
+KEEP, VDIOTKC. ********************************************************
      INTEGER MAXTK, MXVTK
      PARAMETER (MAXTK= 500)           ! Maximum number of tracks
      PARAMETER (MXVTK= 200)           ! Storage allocated per track

      INTEGER NVDTK                    ! Number of tracks this event
      REAL     VDTK(MXVTK,MAXTK)       ! Floating-point track data
      INTEGER IVDTK(MXVTK,MAXTK)       ! Integer track data
*
      EQUIVALENCE  (IVDTK, VDTK)       ! Both stored in the same array
*
      COMMON /VDIOTK/ NVDTK, VDTK      ! Everybody should look at this
```

Important Variables:

`INTEGER   NVDTD` Total number of VD hits in the event

INTEGER	NVDTDX	Number of $R\Phi$ VD hits in the event
INTEGER	NVDTDZ	Number of z VD hits in the event
REAL	VDTD(*,*)	Real variables for each VD hit (see below)
INTEGER	IVDTD(*,*)	Integer variables for each VD hit (see below)
INTEGER	NVDTK	Number of tracks in the event
REAL	VDTK(*,*)	Real variables for each track (see below)
INTEGER	IVDTK(*,*)	Integer variables for each track (see below)

Usage Notes: These arrays are filled by VDRDST (section 11.3). Note, for 94 data onwards, where the VD contains double-sided detectors, the $R\Phi$ hits are stored in the arrays ahead of the z hits.
For every VD hit I, the information for the hit is stored in VDTD(J,I) or IVDTD(J,I), where J depends on the information required. The values of J are best used through the set of parameters in the keep sequence VDIOTDP. For example, to access the x-local value for the VD hit I, then use VDTD(IVHXLC,I). See VDKEEP.CAR for details of these parameters.
Similarly, the VD-related track information is stored for track I in VDTK(J,I) or IVDTK(J,I), where the parameters for J are defined in keep VDIOTKP. Again, see VDKEEP.CAR for details.
The contents of these commons can be printed out using the routines VDPRHT and VDPRTR (section 11.10).
Persons Responsible: T. Adye, M. Karlsson.

3 Standard Delphi Initialisation

3.1 DSTBEG, DSTEVT, DSTEND

Function: Sets up variables for correct use of `DSTANA` routines.
Source Code: Decks `DSTBEG, DSTEVT, DSTEND`, Patch `UTILITY`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE DSTBEG
                    =====
SUBROUTINE DSTEVT(IBITS)

SUBROUTINE DSTEND
```

Calling Arguments:

Input argument

`INTEGER`	`IBITS`	Bitted word allowing a choice for the initialisation performed. Should normally be zero.

Usage Notes: `DSTBEG` should be called after Zebra initialisation, but before the loop over events is performed. It sets up the constants in `/PXCONS/` (see 2.6), initialises `VDCLAP` and defines the link area `LVECP` in `VECSUB` (see 5).
`DSTEVT` should be called for every event before any analysis is done. It sets up the event-by-event constants in `/PXCONS/`. `IBITS` controls which initialisation is done, but unless the user is an expert, it is dangerous to set this variable non-zero.
`DSTEND` is to be called after all events have been processed. Currently, it only calls `DSTQIE` (see 15.1).
Person Responsible: P. Dauncey

4 Pilot Record and DST Data Links

The Delphi data structure is based around the CERN Zebra structure. The data are stored in a large array (in common /UXCOM/, see 2.4) and "links" (pointers) to the position of the start of the data in this array are defined for getting the data required.

Because Zebra can move the position of the data stored in the array, to make room for additions, histograms etc, then a link cannot be considered to be a fixed quantity, but may change within an event. This means that if a link is stored in a local variable, then it may become obselete without any warning. This can be overcome by telling Zebra that certain memory areas are used for links and so if the data are shifted in memory, then the links are automatically updated also. These defined memory areas must be in a common and the definition is done through a call to the Zebra routine MZLINK. All this means that when using links, it is much safer to store them in such a defined array unless they are to be used immediately. The VECSUB package (section 5) has an array for links defined in common /PUCLLL/ (section 2.7). Any other defined arrays needed must be initialised by the user.

The routines below are provided for convenience of getting links to the different parts of the DST data structure. None of them store the links automatically in the /PUCLLL/ array, but this can be done using DSTVEC (section 5.2) if required for the PA bank links themselves. Many of these routines are from the PHDST package, and the following is written by the author, V. Perevoztchikov;

DELPHI DST format is complicated enough. ZEBRA structure of DST DATA or PILOT record is simple and clear, but it is not very easy to understand the internal structure of banks or blocklets. Banks or blocklets contain blocks, those blocks contain others blocks and so on. Even if the user knows this structure very well, his code becomes very sophisticated, due to the complexity of structure. PHDST-TOOLS gives the user the possibility of simplifying and standardizing his code. That is important if his collaborators are to understand it. The most useful utilities are IPHPIC, PHRTY and LPHPA.

4.1 Pilot Record Routines

4.1.1 PHIPID

Function: Initialisation for the PHDST pilot tools.
Source Code: Deck `PHIPID`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE PHIPID
```

Calling Arguments: None
Usage Notes: `PHIPID` must be called once per job before using any of the following PHDST tools for the pilot record which reference blocklets through their character names. This call is done automatically within a normal PHDST job.
Person Responsible: N. Smirnov

4.1.2 IPHPIX

Function: Finds the next blocklet address in the pilot record.
Source Code: Deck `IPHPIX`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                             ======
INTEGER FUNCTION IPHPIX(IACURR)
                 ======
```

Calling Arguments:

Input argument

`INTEGER`	`IACURR`	Address of current blocklet in pilot (-1 for first blocklet)

Output

`INTEGER`	`IPHPIX`	Address of next blocklet in pilot -1 = end of the blocklets, -13 = some contradiction encountered.

Usage Notes: `IPHPIX` gives the next blocklet address in `IUPILT` array, using the current blocklet address . The address is defined in such way that `IUPILT(IADDRESS+1)` = Number of words in blocklet.
The address of the first blocklet (DAS blocklet) is zero.
Person Responsible: N. Smirnov

4.1.3 IPHPIL

Function: Search blocklet address in PILOT record by integer blocklet type.
Source Code: Deck `IPHPIL`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                              == =====
INTEGER FUNCTION IPHPIL(ID,IDSUB)
                     ======
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Blocklet type
`INTEGER`	`IDSUB`	Blocklet sub-type

Output

`INTEGER`	`IPHPIL`	Address of the blocklet found in pilot

Person Responsible: N. Smirnov

4.1.4 IPHPIC

Function: Search blocklet address in PILOT record by character blocklet type.
Source Code: Deck `IPHPIC`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                              ===== ======
INTEGER FUNCTION IPHPIC(CTYPE,IDUMMY)
                 ======
```

Calling Arguments:

Input arguments

`CHARACTER*4`	`CTYPE`	Blocklet type
`INTEGER`	`IDUMMY`	Dummy argument, not used

Output

`INTEGER`	`IPHPIC`	Address of the blocklet found in pilot

Usage Notes: This function is almost the same as `IPHPIL` (section 4.1.3), but instead of numerical blocklet type mnemonical names which are logically connected to numerical blocklet types are used. For more details, see `PHPID` (section 4.1.5) and `IPHPID` (section 4.1.6).
To use this routine, the routine `PHIPID` must be called for initialisation.
Person Responsible: N. Smirnov

4.1.5 PHPID

Function: Converting integer pilot blocklet type to character type.
Source Code: Deck PHPID, Patch DSTOOLS, Cards PHDST, Library PHDST, Disk PHY.
Calling Sequence:

```
                           == =====
CHARACTER*4 FUNCTION PHPID(ID,IDSUB)
                     =====
```

Calling Arguments:

Input argument

INTEGER	ID	Blocklet type
INTEGER	IDSUB	Blocklet sub-type

Output

CHARACTER*4	PHPID	Character blocklet type

Usage Notes: Converts INTEGER pilot blocklet type (2nd word of the blocklet) to a CHARACTER*4 identifer.
The types are:

```
00=DAS, 01=T1,  02=T2,  03=T3, 04=T4,  05=ONL5,06=ONL6,07=IDEN
08=DANA,09=DPHA,10=DJET,11=LUM,12=TRIG,13=0013,14=0014,15=DTAG
```

To use this routine, the routine PHIPID must be called for initialisation.
Person Responsible: N. Smirnov

4.1.6 IPHPID

Function: Converting character pilot blocklet type to integer.
Source Code: Deck `IPHPID`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                  =====
SUBROUTINE IPHPID(CTYPE,ID,IDSUB)
                        == =====
```

Calling Arguments:

Input argument

`CHARACTER*4`	`CTYPE`	Character blocklet type

Output

`INTEGER`	`ID`	Blocklet type
`INTEGER`	`IDSUB`	Blocklet sub-type

Usage Notes: Converts `CHARACTER*4` pilot blocklet type to `INTEGER` type.
The types are:

```
00=DAS, 01=T1,  02=T2,  03=T3, 04=T4,  05=ONL5,06=ONL6,07=IDEN
08=DANA,09=DPHA,10=DJET,11=LUM,12=TRIG,13=0013,14=0014,15=DTAG
```

To use this routine, the routine `PHIPID` must be called for initialisation.
Person Responsible: N. Smirnov

4.1.7 PHRTY

Function: Get character Record Type
Source Code: Deck PHRTY, Patch DSTOOLS, Cards PHDST, Library PHDST, Disk PHY.
Calling Sequence:

```
CHARACTER*4 FUNCTION PHRTY()
                     =====
```

Calling Arguments:

Output
`CHARACTER*4` `PHRTY` Record type

Usage Notes: Converts INTEGER pilot record type (IPIRTY=4 word of IUPILT) to CHARACTER*4 type.
List of record types is: 1=RAW,2=SIM,3=TAN,4=INT,5=DST or SDST.
Input is via common /PXCHDR/ array IUPILT.
To use this routine, the routine PHIPID must be called for initialisation.
Person Responsible: N. Smirnov

4.1.8 KBLCKAD

Function: Find the address in Pilot record of blocklet ID,ISUB.
Source Code: Deck `KBLCKAD`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                                == ====
INTEGER FUNCTION KBLCKAD(ID,ISUB)
                     =======
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Identifier of the blocklet
`INTEGER`	`ISUB`	Subidentifier of the blocklet

Output

`INTEGER`	`KBLCKAD`	Address-1 of the blocklet
		-1 Inconsistant blocklet length in `IUPILT`
		-2 No blocklet found with such identifiers

Usage Notes: No initialisation is required for this routine.
Person Responsible: L. Mirabito

4.2 DST Data Link Routines

4.2.1 PHIPAX

Function: Initialisation for the PHDST links tools.
Source Code: Deck `PHIPAX`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE PHIPAX
```

Calling Arguments: None
Usage Notes: `PHIPAX` must be called once per job before using any of the following PHDST tools for the links which reference extra modules through their character names. This call is done automatically within a normal PHDST job.
Person Responsible: N. Smirnov

4.2.2 LPHPAN, LPHPAB

Function: Get the address of the extra modules in the PA bank.
Source Code: Deck `LPHPAN`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                             === ===
INTEGER FUNCTION LPHPAN(LPA,NUM)
                    ======
                             === ===
INTEGER FUNCTION LPHPAB(NUM,LPA)
                    ======
```

Calling Arguments:

Input arguments

`INTEGER`	`LPA`	Address of PA bank
`INTEGER`	`NUM`	Extra module position in list (for LPHPAN) or identifier (for LPHPAB)

Output

`INTEGER`	`LPHPAN`	Address of NUMth module
`INTEGER`	`LPHPAB`	Address of module NUM

Usage Notes: `LPHPAN` finds the address of the NUMth module in the order they appear in the PA bank, irrespective of which module it is. `LPHPAB` finds the address of the module of number NUM. Immediately after using `LPHPAN` (but NOT `LPHPAB`), the length (number of words) of the module can be obtained with a call to a function, `IPHREQ(1)`.
Person Responsible: N. Smirnov

4.2.3 LPHPA

Function: Get the address of the nth (SUB(SUB))module in the PA bank.
Source Code: Deck `LPHPA`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                           === === ===
INTEGER FUNCTION LPHPA(IDD,LPA,NUM)
                 =====
```

Calling Arguments:

Input arguments

`CHARACTER*(*)`	`IDD`	Character string identifying type of information
`INTEGER`	`LPA`	Address of PA bank or “parent” module
`INTEGER`	`NUM`	Module number

Output

`INTEGER`	`LPHPA`	Address of NUMth module

Usage Notes: This is the main function of PHTOOLS. The types of extra modules are:

```
LXRA = LPHPA ('MAIN',LPA,0)
LXRA = LPHPA ('CCAL',LPA,0)
LXRA = LPHPA ('EMCA',LPA,0)
LXRA = LPHPA ('HCAL',LPA,0)
LXRA = LPHPA ('MU'  ,LPA,0)
LXRA = LPHPA ('EL'  ,LPA,0)
LXRA = LPHPA ('MRIC',LPA,0)
LXRA = LPHPA ('MTPC',LPA,0)
LXRA = LPHPA ('TRAC',LPA,0)
LXRA = LPHPA ('TOF' ,LPA,0)
LXRA = LPHPA ('TDHA',LPA,0)
LXRA = LPHPA ('TEID',LPA,0)
LXRA = LPHPA ('TETP',LPA,0)
LXRA = LPHPA ('TEOD',LPA,0)
LXRA = LPHPA ('TEFA',LPA,0)
LXRA = LPHPA ('TEFB',LPA,0)
LXRA = LPHPA ('TDID',LPA,0)
LXRA = LPHPA ('EMNC',LPA,0)
LXRA = LPHPA ('HCNC',LPA,0)
LXRA = LPHPA ('MUID',LPA,0)
LXRA = LPHPA ('ELID',LPA,0)
LXRA = LPHPA ('HAID',LPA,0)
LXRA = LPHPA ('ETRA',LPA,0) ????
LSHOW = LPHPA ('CCAL.SHOWER',LCCAL,NSHOW)
LSHOW = LPHPA ('EMCA.SHOWER',LEMCA,NSHOW)
LSHOW = LPHPA ('HCAL.SHOWER',LHCAL,NSHOW)
```

```
LLAY = LPHPA ('CCAL.SHOWER.LAYER',LSHOW,NLAY)
LLAY = LPHPA ('EMCA.SHOWER.LAYER',LSHOW,NLAY)
LLAY = LPHPA ('HCAL.SHOWER.LAYER',LSHOW,NLAY)
LLAY = LPHPA ('MU.LAYER',LMU,NLAY)
```

Immediately after using `LPHPA`, the length (number of words) of the module can be obtained with a call to a function, `IPHREQ(1)`. Similarly, the order number of the extra module is given by `IPHREQ(2)`, where the main module gives 1.
To use this routine, the routine `PHIPAX` must be called for initialisation.
Person Responsible: N. Smirnov

4.2.4 PHXID

Function: Converting Xtra modules ID to names
Source Code: Deck `PHXID`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                          ==
CHARACTER*4 FUNCTION PHXID(ID)
                     =====
```

Calling Arguments:

Input argument
`INTEGER` `ID` Number of blocklet

Output
`CHARACTER*4` `PHXID` Name of blocklet

Usage Notes: Accordance between Xtra modules (blocklets) names and ID;

```
  MODULE              NAME       ROUTINE   DST ID    COMMENTS
  Combined calo.     'CCAL'      PXCCAL     1.
  Elec-mag calo.     'EMCA'      PXEMCA     2. (Barrel,end-caps,sat)
  Hadronic calo.     'HCAL'      PXHCAL     3. (Barrel,end-caps)
  Muon identif.      'MU  '      PXMU       4.
  Elect identif.     'EL  '      PXel       5.
  Mass id RICH       'MRIC'      PXMRIC     6.
  mass id TPC        'MTPC'      PXMTPC     7.
  Trac info.         'TRAC'      PXTRAC     8.
  Time of flight     'TOF '      PXTOF      9.
  TD from HAC        'TDHA'      PXTDHA     11.
  TE from ID         'TEID'      PXTEAM(4)  12.
  TE from TPC        'TETP'      PXTEAM(5)  13.
  TE from OD         'TEOD'      PXTEAM(8)  14.
  TE from FCA        'TEFA'      PXTEAM(20) 15.
  TE from FCB        'TEFB'      PXTEAM(25) 16.
  TD from ID         'TDID'      PXTDID     17.
  SAT info           'SAT '      PXSAT      18.
  EM Cal Info        'EMNC'                 22.
  Had  Cal Info      'HCNC'                 23.
  Muon id info       'MUID'                 24.
  Electron id info   'ELID'                 25.
  Hadron id info     'HAID'                 26.
  Electron track fit 'ETRA'                 28.
```

To use this routine, the routine `PHIPAX` must be called for initialisation.
Person Responsible: N. Smirnov

4.2.5 IPHXID

Function: Converting Xtra modules names to ID.
Source Code: Deck `IPHXID`, Patch `DSTOOLS`, Cards `PHDST`, Library `PHDST`, Disk `PHY`.
Calling Sequence:

```
                             ===
INTEGER FUNCTION IPHXID(CID)
                 ======
```

Calling Arguments:

Input argument

`CHARACTER*(*)`	`CID`	Name of blocklet

Output

`INTEGER`	`IPHXID`	Number of blocklet

Usage Notes: To use this routine, the routine `PHIPAX` must be called for initialisation.
Person Responsible: N. Smirnov

4.2.6 DSTLPA

Function: Finds all links to PA's in the DST hanging from `LDTOP-1`.
Source Code: Deck `DSTLPA`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                    ======
SUBROUTINE DSTLPA(NPAMAX,NDSTPA,LDSTPA)
                            ====== ======
```

Calling Arguments:

Input argument

`INTEGER`	`NPAMAX`	Maximum number of PA's to be stored = size of `LDSTPA` array

Output arguments

`INTEGER`	`NDSTPA`	Number of PA's found
`INTEGER`	`LDSTPA(NPAMAX)`	Links to PA's

Usage Notes: This routine returns all the PA's found in the order in which they occur on the DST. It is often better to use routine DSTPAS (section 4.2.7) which sorts the PA's by charge and momentum/energy.
Person Responsible: P. Dauncey

4.2.7 DSTPAS

Function: Finds all links to PA's in the DST hanging from `LDTOP-1` and sorts them; firstly all charged tracks in order of absolute momentum and then neutrals in order of energy.
Source Code: Deck `DSTPAS`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                             ======
SUBROUTINE DSTPAS(NPAMAX,NPACHG,NPANEU,LDSTPA)
                                  ====== ====== ======
```

Calling Arguments:

Input argument

`INTEGER`	`NPAMAX`	Maximum number of PA's to be stored = size of `LDSTPA` array

Output arguments

`INTEGER`	`NPACHG`	Number of charged PA's found
`INTEGER`	`NPANEU`	Number of neutral PA's found
`INTEGER`	`LDSTPA(NPAMAX)`	Links to PA's

Usage Notes: This routine returns all the PA's found ordered by charge and momentum/energy. The first `NPACHG` links are to charged tracks, so that `LDSTPA(1)` is the link to the highest momentum charged track and `LDSTPA(NPACHG)` is the link to the lowest momentum charged track. In addition, `LDSTPA(NPACHG+1)` is the link to the highest energy neutral track and `LDSTPA(NPACHG+NPANEU)` is the link to the lowest energy neutral track.
Person Responsible: P. Dauncey

4.2.8 DPABLK

Function: Finds links to blocklets for a PA bank.
Source Code: Deck DPABLK, Patch DSTIO, Cards DSTANA, Library DSTANA, Disk PHY.
Calling Sequence:

```
                    ======
SUBROUTINE DPABLK(LDSTPA,LBLCK0,LBLCKN,NBLCKN)
                            ====== ====== ======
```

Calling Arguments:

Input argument

INTEGER	LDSTPA	Link to PA bank Output arguments
INTEGER	LBLCK0	Link to start of blocklet 0 (standard blocklet)
INTEGER	LBLCKN(*)	Links to other blocklets
INTEGER	NBLCKN(*)	Lengths of other blocklets

Usage Notes: The standard blocklet length is always 27 words.
The link points to the first word of the blocklet, so that for blocklet I, Q(LBLCKN(I))=I.0 is the first word. Also, Q(LBLCKN(I)+NBLCKN(I)-1) is the last word of the blocklet.
The arrays LBLCKN and NBLCKN must be long enough to accomodate the highest PA extra module number. Currently, this is 30 on the Short-DST, although private analyses use numbers up to 40. To be sure of not overwriting the arrays used, there is a keep sequence MXBLCK in DSTANA which defines the arrays using a parameter MXBLCK=40 which will be increased as necessary. Using this ensures the code will be automatically maintained.
Person Responsible: P. Dauncey

4.2.9 DPAB12

Function: Finds links to ID TER's for an PA blocklet 12.
Source Code: Deck `DPAB12`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                       ====== ======
SUBROUTINE DPAB12(LBLK12,NBLK12,NTE,LTER,NTER)
                                         === ==== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`LBLK12`	Link to PA blocklet 12
`INTEGER`	`NBLK12`	Length of PA blocklet 12

Output arguments

`INTEGER`	`NTE`	Number of ID TER's found in blocklet 12
`INTEGER`	`LTER(3)`	Links to ID TER's
`INTEGER`	`NTER(3)`	Lengths of ID TER's

Usage Notes: The link points to the first word of the TER, so `Q(LTER(1))` is the measurement code of the first TER, `Q(LTER(1)+1)` is the radius of the first TER etc.
Person Responsible: P. Dauncey

5 VECSUB Interface and Routines

VECSUB is an interface between the Delphi data and the LUND routines for calculating thrust, sphericity, jets etc. It also provides a set of routines to manipulate the momentum vectors. The interface is all based around the common, /PUCPPP/ (section 2.7), which contains an array VECP(10,*) (equivalenced to array IVECP(10,*)). In addition, there is another common, /PUCLLL/ (section 2.7), which contains links to the PA banks in the DST structure. This makes it easy to find more information about any track which is stored in the VECP array. The common /PUCLLL/ is a Zebra link area and so the links are protected from shifts of the data in memory.

5.1 PUCINI

Function: Initialises link area LVECP in common /PUCLLL/ as Zebra reference links.
Source Code: Deck PUCINI, Patch VECSUB, Cards VECSUB72, Library DSTANA, Disk PHY.
Calling Sequence:

```
SUBROUTINE PUCINI
```

Calling Arguments: None
Usage Notes: DSTVEC (section 5.2) calls PUCINI on its first call. If the user does not call DSTVEC, then it may be necessary to call PUCINI explicitly. It is needed if the link area LVECP is used and calls are made to Zebra routines which may move the links. Note that the Delphi standard initialisation routine DSTBEG (section 3.1) calls PUCINI anyway.
This routine contains a dummy call (when the operating system allows it) to blockdata LUDATA to ensure it is loaded. This blockdata contains all the JETSET default parameters and is needed for correct functioning of PUTHRU and the jet-finding routines below.
Person Responsible: P. Dauncey

5.2 DSTVEC

Function: Fills all reconstructed PA banks into the `VECP` array and links into the `LVECP` array.
Source Code: Deck `DSTVEC`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE DSTVEC(NPACHR,NPANEU)
                  ====== ======
```

Calling Arguments:

Output arguments

`INTEGER`	`NPACHR`	Number of charged PA's put into VECP
`INTEGER`	`NPANEU`	Number of neutral PA's put into VECP

Usage Notes: The PA's are sorted by charge and energy/momentum.
On the first call to this routine, PUCINI (section 5.1) is called to initialise the link array `LVECP`.
Person Responsible: P. Dauncey

5.3 DPAVEC

Function: Fills a single PA bank into a VECP-like array.
Source Code: Deck DPAVEC, Patch DSTIO, Cards DSTANA, Library DSTANA, Disk PHY.
Calling Sequence:

```
                  ======
SUBROUTINE DPAVEC(LDSTPA,VECP)
                         ====
```

Calling Arguments:

Input argument

INTEGER	LDSTPA	Link to PA bank

Output argument

REAL	VECP(10)	Array containing VECP-like information

Usage Notes: Use DSTVEC (section 5.2) to get all PA banks in the event.
Person Responsible: P. Dauncey

5.4 ADD3X, SUB3X, CROSSX, DOTX, SMULX, NORZX, LENGTX, ANGX, PARPEX, PUNITX, DEFSX, TRACX, TRAPX

Function: Performs 3-vector operations on momenta in `VECP` array.
Source Code: Deck `VECSUB`, Patch `VECSUB`, Cards `VECSUB72`, Library `DSTANA`.
Calling Sequence:

```
                 = = =
SUBROUTINE ADD3X(I,J,K)

                 = = =
SUBROUTINE SUB3X(I,J,K)

                  = = =
SUBROUTINE CROSSX(I,J,K)

                = =
SUBROUTINE DOTX(I,J,U)
                    =
                 = = =
SUBROUTINE SMULX(I,K,S)

                 = =
SUBROUTINE NORZX(I,K)

                  =
SUBROUTINE LENGTX(I,U)
                    =
                = =
SUBROUTINE ANGX(I,J,U,V)
                    = =
                  = =
SUBROUTINE PARPEX(I,J,U,V)
                      = =
                  = =
SUBROUTINE PUNITX(K,L)

                 = = =
SUBROUTINE DEFSX(I,J,K)

                 = = =
SUBROUTINE TRACX(I,J,K)

                 = =
SUBROUTINE TRAPX(I,J,U,V,W)
                     = = =
```

Calling Arguments:

Input arguments

INTEGER	I,J	Track numbers in VECP array
INTEGER	L	Component of vector (x,y,z = 1,2,3)
INTEGER	K	Track number for result of vector calculation
REAL	S	Scalar input variable

Output arguments

REAL	U,V,W	Results of scalar calculation

Usage Notes: All these routines only use and/or affect the first three components.
ADD3X gives $\mathbf{p}_k = \mathbf{p}_i + \mathbf{p}_j$.
SUB3X gives $\mathbf{p}_k = \mathbf{p}_i - \mathbf{p}_j$.
CROSSX gives $\mathbf{p}_k = \mathbf{p}_i \times \mathbf{p}_j$.
DOTX gives U = $\mathbf{p}_i . \mathbf{p}_j$.
SMULX gives $\mathbf{p}_k$ = S $\mathbf{p}_i$.
NORZX gives $\mathbf{p}_k = \mathbf{p}_i / \mid \mathbf{p}_i \mid$.
LENGTX gives U = $\mid \mathbf{p}_i \mid$.
ANGX gives U = $\mathbf{p}_i . \mathbf{p}_j / \mid \mathbf{p}_i \mid\mid \mathbf{p}_j \mid, V = cos^{-1}(U)$.
PARPEX gives U = component of $\mathbf{p}_i$ parallel to $\mathbf{p}_j$, V = component of $\mathbf{p}_i$ perpendicular to $\mathbf{p}_j$.
PUNITX gives unit vector $\mathbf{p}_k$ along axis L (= 1,2,3).
DEFSX gives unit vectors $\mathbf{p}_{k+2}$ along $\mathbf{p}_i$, $\mathbf{p}_k$ along $\mathbf{p}_i \times \mathbf{p}_j$ and $\mathbf{p}_{k+1}$ along $\mathbf{p}_{k+2} \times \mathbf{p}_k$.
TRACX gives vector $\mathbf{p}_k = \mathbf{p}_i$ transformed to the rotated system defined by the three axes $\mathbf{p}_j$, $\mathbf{p}_{j+1}$ and $\mathbf{p}_{j+2}$.
TRAPX gives the components of $\mathbf{p}_i$ transformed to polar coordinates defined by the three axes $\mathbf{p}_j$, $\mathbf{p}_{j+1}$ and $\mathbf{p}_{j+2}$. U = $\mid \mathbf{p}_i \mid, V = \cos\theta$ and $W = \phi$.
Person Responsible: Y. Sacquin

5.5 ADDX, SUBX, DOT4X, IMPUX, LORX

Function: Performs 4-vector operations on momenta in `VECP` array.
Source Code: Deck `VECSUB`, Patch `VECSUB`, Cards `VECSUB72`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                = = =
SUBROUTINE ADDX(I,J,K)

                = = =
SUBROUTINE SUBX(I,J,K)

                 = =
SUBROUTINE DOT4X(I,J,U)
                     =
                 = =
SUBROUTINE IMPUX(I,J,U)
                     =
                = = =
SUBROUTINE LORX(I,J,K)
```

Calling Arguments:

Input arguments

`INTEGER`	`I,J`	Track numbers in `VECP` array
`INTEGER`	`K`	Track number for result of vector calculation

Output argument

`REAL`	`U`	Result of scalar calculation

Usage Notes: All these routines only affect the first five components.
`ADDX` gives $p_{\mu k} = p_{\mu i} + p_{\mu j}$.
`SUBX` gives $p_{\mu k} = p_{\mu i} - p_{\mu j}$.
`DOT4X` gives $U = p_{\mu i} p_j^{\mu}$.
`IMPUX` gives $U = -(p_{\mu i} - p_{\mu j})(p_i^{\mu} - p_j^{\mu})$.
`LORX` gives $p_{\mu k}$ as p_{μ_i} transformed to the rest frame of $p_{\mu j}$.
Person Responsible: Y. Sacquin

5.6 ADD7X, SUB7X, LOR7X

Function: Perform 4-vector operations on momenta in `VECP` array and additionally calculate the mass, charge and total momentum.
Source Code: Deck `VECSUB`, Patch `VECSUB`, Cards `VECSUB72`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                 = = =
SUBROUTINE ADD7X(I,J,K)

                 = = =
SUBROUTINE SUB7X(I,J,K)

                 = = =
SUBROUTINE LOR7X(I,J,K)
```

Calling Arguments:

Input arguments

`INTEGER`	`I,J`	Track numbers in `VECP` array
`INTEGER`	`K`	Track number for result of vector calculation

Usage Notes: All these routines affect the first seven components, i.e. the values of the mass, charge and total absolute momentum are also calculated.
`ADD7X` gives $p_{\mu k} = p_{\mu i} + p_{\mu j}$.
`SUB7X` gives $p_{\mu k} = p_{\mu i} - p_{\mu j}$.
`LOR7X` gives $p_{\mu k}$ as p_{μ_i} transformed to the rest frame of $p_{\mu j}$.
Person Responsible: Y. Sacquin

5.7 PCOPX, PEXCX, PZERX, PWRTX, PUORDR

Function: Performs handling operations on all 10 components of the VECP array.
Source Code: Deck VECSUB, Patch VECSUB, Cards VECSUB72, Library DSTANA.
Calling Sequence:

```
                 = = =
SUBROUTINE PCOPX(I,J,K)
                 = =
SUBROUTINE PEXCX(I,J)
                 = =
SUBROUTINE PZERX(I,J)
                 = = =
SUBROUTINE PWRTX(I,J,M)
                  = = = = =====
SUBROUTINE PUORDR(I,J,L,K,IFLAG)
```

Calling Arguments:

Input arguments

INTEGER	I,J,K	Track numbers in VECP array
INTEGER	M	Number of calls for printout
INTEGER	L	Component of VECP vector
INTEGER	IFLAG	Switch for ordering; $+1$ increasing order, -1 decreasing order

Usage Notes: These routines manipulate the LVECP tracks links as well as the VECP vectors.
PCOPX gives $p_k = p_i$, $p_{k+1} = p_{i+1}$ ... $p_{k-j+1} = p_j$.
PEXCX gives $p_i = p_j$ and $p_j = p_i$.
PZERX zeros p_i to p_j.
PWRTX prints out a table to LUN=6 of p_i to p_j for the first M calls (cf VECPRT, section 5.13).
PUORDR orders p_i to p_j according to the value of component L. The ordered vectors are stored in p_k, p_{k+1} ... The ordering is in increasing value if IFLAG=+1 and decreasing for IFLAG=−1.
Person Responsible: Y. Sacquin

5.8 PUSPHR, PUSPHE

Function: Calculates the sphericity and axes from the momenta in the `VECP` array.
Source Code: Deck `PUSPHR`, Patch `VECSUB`, Cards `VECSUB72`, Library `DSTANA`.
Calling Sequence:

```
                  ==== ==== =====        =====
SUBROUTINE PUSPHR(ITRK,NTRK,IAXES,SPH,APL,POWER)
                                  === ===
                  ==== ==== =====        =====
SUBROUTINE PUSPHE(ITRK,NTRK,IAXES,SPH,APL,POWER,NHEM1,NHEM2,EMAX1,EMAX2)
                                  === ===       ===== ===== ===== =====
```

Calling Arguments:

Input arguments

`INTEGER`	`ITRK`	Track number of first particle to be used in `VECP`
`INTEGER`	`NTRK`	Number of tracks to be used
`INTEGER`	`IAXES`	Position in `VECP` of three orthogonal axes (`IAXES`, `IAXES+1` and `IAXES+2`)
`REAL`	`POWER`	Power of momentum weighting to be used (= 2.0 normally)

Output arguments

`REAL`	`SPH`	Value of sphericity
`REAL`	`APL`	Value of aplanarity
`INTEGER`	`NHEM1`	Number of particles in hemisphere 1
`INTEGER`	`NHEM2`	Number of particles in hemisphere 2
`REAL`	`EMAX1`	Energy of fastest particle in hemisphere 1
`REAL`	`EMAX2`	Energy of fastest particle in hemisphere 2

Usage Notes: The tracks in the `VECP` array are supposed to be labelled as being associated with hemisphere 1 or 2 by the value of `IVECP(10,I)`, which is incremented by `100*H`. Note, multiple calls to thrust and sphericity routines will continually update this number and so make it meaningless. THIS DOES NOT APPEAR TO BE IMPLEMENTED (YET).
Person Responsible: Y. Sacquin

5.9 PUTHRU

Function: Calculates the thrust and axes from the momenta in the VECP array.
Source Code: Deck PUTHRU, Patch VECSUB, Cards VECSUB72, Library DSTANA, Disk PHY.
Calling Sequence:

```
                  ==== ==== =====          =====
SUBROUTINE PUTHRU(ITRK,NTRK,IAXES,THR,OBL,POWER)
                                  === ===
```

Calling Arguments:

Input arguments

INTEGER	ITRK	Track number of first particle to be used in VECP
INTEGER	NTRK	Number of tracks to be used
INTEGER	IAXES	Position in VECP of three orthogonal axes (IAXES, IAXES+1 and IAXES+2)
REAL	POWER	Power of momentum weighting to be used (= 1.0 normally)

Output arguments

REAL	THR	Value of thrust
REAL	OBL	Value of oblateness

Usage Notes: The tracks in the VECP array are supposed to be labelled as being associated with hemisphere 1 or 2 by the value of IVECP(10,I), which is incremented by 100*H. Note, multiple calls to thrust and sphericity routines will continually update this number and so make it meaningless. THIS DOES NOT APPEAR TO BE IMPLEMENTED (YET).
Person Responsible: Y. Sacquin

5.10 PUCLUS

Function: Calculates jet axes using a cluster method from the momenta in the VECP array.
Source Code: Deck PUCLUS, Patch VECSUB, Cards VECSUB72, Library DSTANA, Disk PHY.
Calling Sequence:

```
                  ==== ==== =====
SUBROUTINE PUCLUS(ITRK,NTRK,IAXES,NAXES,TGEN,DMIN)
                                  ===== ==== ====
```

Calling Arguments:

Input arguments

INTEGER	ITRK	Track number of first particle to be used in VECP
INTEGER	NTRK	Number of tracks to be used
INTEGER	IAXES	Position in VECP of jet axes (IAXES, IAXES+1 etc)

Output arguments

INTEGER	NAXES	Number of jet axes found
REAL	TGEN	Value of generalised thrust
REAL	DMIN	Minimum "distance" between jets in cluster space

Usage Notes: The tracks in the VECP array are labelled as being associated with jet N by the value of IVECP(10,I), which is incremented by 1000*n. Note, multiple calls to jet-finding routines will continually update this number and so make it meaningless.
This routine calls the JETSET routine LUCLUS, which is controlled by two parameters, PARU(43) and PARU(44), in the common /LUDAT1/. Their default values are set in blockdata LUDATA, which must be loaded. This is ensured by a dummy call to LUDATA in PUCINI (section 5.1).
Person Responsible: Y. Sacquin

5.11 PUYCLU

Function: Calculates jet axes using an invariant mass method (JADE) from the momenta in the VECP array.
Source Code: Deck PUYCLU, Patch VECSUB, Cards VECSUB72, Library DSTANA, Disk PHY.
Calling Sequence:

```
                             ==== ==== =====        ==== ======
SUBROUTINE PUYCLU(ITRK,NTRK,IAXES,NAXES,YMIN,JETFIX,MINTR,YMINCT)
                                        =====            ===== ======
```

Calling Arguments:

Input arguments

INTEGER	ITRK	Track number of first particle to be used in VECP
INTEGER	NTRK	Number of tracks to be used
INTEGER	IAXES	Position in VECP of jet axes (IAXES, IAXES+1 etc)
REAL	YMIN	Scaled mass cutoff for jets
INTEGER	JETFIX	Stop jet-finding after JETSET jets (default = 0 , no limits)

Output arguments

INTEGER	NAXES	Number of jet axes found
REAL	YMINCT	Last scaled invariant mass value which did not pass the YMIN cut

Usage Notes: The tracks in the VECP array are labelled as being associated with jet N by the value of IVECP(10,I), which is incremented by 1000*n. Note, multiple calls to jet-finding routines will continually update this number and so make it meaningless.
Meaning of the scaled mass cut off :

- ABS(YMIN) > 1. the value directly is taken as mass cutoff
- ABS(YMIN) < 1. the value M=SQRT(YMIN*EVIS**2) is taken as mass cutoff with EVIS = total energy of the particles
- YMIN > 0. the mass is given by M**2=2*E1*E2*(1-COSTHETA)
- YMIN < 0. the mass is given by M**2=E**2-P**2

Person Responsible: Y. Sacquin

5.12 PUDCLU

Function: Calculates jet axes using an invariant mass method (DURHAM) from the momenta in the VECP array.
Source Code: Deck PUDCLU, Patch VECSUB, Cards VECSUB72, Library DSTANA, Disk PHY.
Calling Sequence:

```
                  ==== ==== =====      ==== ======
SUBROUTINE PUDCLU(ITRK,NTRK,IAXES,NAXES,YMIN,JETFIX,MINTR,YMINCT)
                                  =====            ===== ======
```

Calling Arguments:

Input arguments

INTEGER	ITRK	Track number of first particle to be used in VECP
INTEGER	NTRK	Number of tracks to be used
INTEGER	IAXES	Position in VECP of jet axes (IAXES, IAXES+1 etc)
REAL	YMIN	Scaled mass cutoff for jets
INTEGER	JETFIX	Stop jet-finding after JETSET jets (default = 0 , no limits)

Output arguments

INTEGER	NAXES	Number of jet axes found
REAL	YMINCT	Last scaled invariant mass value which did not pass the YMIN cut

Usage Notes: The tracks in the VECP array are labelled as being associated with jet N by the value of IVECP(10,I), which is incremented by 1000*n. Note, multiple calls to jet-finding routines will continually update this number and so make it meaningless.
Meaning of the scaled mass cut off :

- For 0. < YMIN < 1. , the value M=SQRT(YMIN*EVIS**2) is taken as mass cutoff with EVIS = total energy of the particles
- For YMIN > 1. YMIN is directly taken as cutoff.

The mass is always given by
M**2 = MAX(0.,2 *min(E1**2,E2**2)*(1.-cos(theta))
Person Responsible: Y. Sacquin

5.13 VECPRT

Function: Prints out a table of the tracks stored in the VECP array.
Source Code: Deck `VECPRT`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                    ====== ====== ======
SUBROUTINE VECPRT(ILUNIT,NVECLO,NVECHI)
```

Calling Arguments:

Input arguments

`INTEGER`	`ILUNIT`	Logical unit number for printout
`INTEGER`	`NVECLO`	First element of VECP to be printed out
`INTEGER`	`NVECHI`	Last element of VECP to be printed out

Usage Notes:
Person Responsible: P. Dauncey

6 Particle Identification Routines

The routines for particle identification fall into four main categories. These are electron and photon identification (by the task group ELECID), muon identification (by the group MUIDENT), RICH information and TPC dE/dx information (both by the HADIDENT group).

6.1 ELPROB, ELPROY

Function: Calculates probabilities for a PA bank to be an electron or a pion.
Source Code: Deck ELPROB, Patch ELPROB, Cards ELPROB, Library DSTANA.
Calling Sequence:

```
                  === =====
SUBROUTINE ELPROB(LPA,XYZBS,ITAG,IVETO)
                            ==== =====
                  ===== === =====
SUBROUTINE ELPROY(CYEAR,LPA,XYZBS,ITAG,IVETO)
                                  ==== =====
```

Calling Arguments:

Input arguments		
INTEGER	LPA	PA bank link
REAL	XYZBS(3)	Beamspot position
CHARACTER*4	CYEAR	Type of data, '91F ','92D ' etc.
Output arguments		
INTEGER	ITAG	Tag quality, 0-3
INTEGER	IVETO	Gamma conversion veto, 0-3

Usage Notes: This routine calls DSFHPC (section 12.4) to fix the HPC data before calculating the probabilities. Hence, the user need not call DSFHPC previously.
The standard routine is ELPROB, which calculates the data type internally using DSTQID (section 15.1). ELPROY is provided for situations when the data type is to be forced to a fixed value.
Persons Responsible: S. Schael

6.2 ELCONV

Function: Calculates tangent radius for a track, where it points back to the origin, so as to identify conversion electrons.
Source Code: Deck ELCONV, Patch ELCONV, Cards ELCONV, Library DSTANA, Disk ELECID.
Calling Sequence:

```
                    === ==== ====
SUBROUTINE ELCONV(LPA,MODE,BMAG,IERR)
                                ====
```

Calling Arguments:

Input arguments

INTEGER	LPA	PA bank link
INTEGER	MODE	Mode flag; 0 = no cuts, >0 = cuts to clean sample
REAL	BMAG	Magnetic field in Tesla

Output argument

INTEGER	IERR	Error flag

Usage Notes:
Persons Responsible: O. Podobrin, S. Schael

6.3 ELSHFI

Function: Performs a longitudinal shower fit to give an improved HPC energy resolution.
Source Code: Deck `ELSHFI`, Patch `ELSHFI`, Cards `ELSHFI`, Library `DSTANA`, Disk `ELECID`.
Calling Sequence:

```
                    ===== ====== ==== =====
SUBROUTINE ELSHFI(IYEAR,PTRACK,ELAY,THETA,CHIQ,EFIT,NFREE)
                                               ==== ==== =====
```

Calling Arguments:

Input arguments		
`INTEGER`	`IYEAR`	Year of data (91 or 92)
`REAL`	`PTRACK`	Momentum of track
`REAL`	`ELAY(10)`	Energy in the 10 HPC layers
`REAL`	`THETA`	θ angle of track
Output arguments		
`REAL`	`CHIQ`	χ^2 from shower fit; < 0 means no fit
`REAL`	`EFIT`	Corrected energy from fit
`INTEGER`	`NFREE`	Number of d.o.f. of fit

Usage Notes:
Persons Responsible: C. Kreuter, S. Schael

6.4 PXPHOT, PXPHOS

Function: Does HPC photon identification.
Source Code: Deck `PXPHOT`, Patch `PXPHOT`, Cards `PXPHOT`, Library `DSTANA`, Disk `ELECID` and Deck `PXPHOS`, Patch `UTILITY`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE PXPHOT
```

```
                          =====
SUBROUTINE PXPHOS(INOUT)
```

Calling Arguments:

Input argument

`INTEGER`	`INOUT`	Flag defining whether input or output

Usage Notes: See Delphi Note XXX??? for details of using these routines.
Persons Responsible: M. Feindt, O. Podobrin, P. Dauncey

6.5 MUFLAG

Function: Identifies tagged muon tracks.
Source Code: Deck `MUFLAG`, Patch `FMULOP` in `*MUFLAG`, Cards `MUFLAG`, Library `DSTANA`, Disk `MUIDENT,TEAM11`.
Calling Sequence:

```
                    =====
SUBROUTINE MUFLAG(ITYPE,NMUS,LMUON)
                           ==== =====
```

Calling Arguments:

Input argument

`INTEGER`	`ITYPE`	Type of tag to be used: 1 = standard, 2 = tight, 3 = loose, 4 = very loose

Output arguments

`INTEGER`	`NMUS`	Number of muons identified with tag (up to 20)
`INTEGER`	`LMUON(20)`	PA identifiers of identified muons

Usage Notes: `MUFLAG` is the official DELPHI muon identification algorithm. For the event `MUFLAG` is called and the number and identifiers of tracks tagged as muons are returned. There are four types of tag available: standard, tight, loose and very loose, with differing efficiency to purity ratios. `MUFLAG` uses the `MUCFIX` module (see 12.5).
An alternative method of calling `MUFLAG`, obtaining the results of all 4 tags at once, is through the routine `MUCAL2` (see 6.6). If the results of more than one tag are of interest, `MUCAL2` is recommended, as it is more economical on CPU than successive calls to `MUFLAG` itself, and overcomes certain otherwise unavoidable consistency problems for Monte Carlo. For further information see DELPHI 93–14 PHYS 263.
Person Responsible: G. Wilkinson

6.6 MUCALL, MUCAL2

Function: Identifies tagged muon tracks.
Source Code: Deck MUFLAG, Patch FMULOP in *MUFLAG, Cards MUFLAG, Library DSTANA, Disk MUIDENT,TEAM11.
Calling Sequence:

```
SUBROUTINE MUCALL(NMUS,LMUONS,NMUT,LMUONT,NMUL,LMUONL)
                  ==== ====== ==== ====== ==== ======

SUBROUTINE MUCAL2(NMUS,LMUONS,NMUT,LMUONT,NMUL,LMUONL,NMUV,LMUONV)
                  ==== ====== ==== ====== ==== ====== ==== ======
```

Calling Arguments:

Output arguments

INTEGER	NMUS	Number of muons identified with standard tag (up to 20)
INTEGER	LMUONS(20)	PA identifiers of standard muons
INTEGER	NMUT	Number of muons identified with tight tag (up to 20)
INTEGER	LMUONT(20)	PA identifiers of tight muons
INTEGER	NMUL	Number of muons identified with loose tag tag (up to 20)
INTEGER	LMUONL(20)	PA identifiers of loose muons
INTEGER	NMUV	Number of muons identified with very loose tag tag (up to 20)
INTEGER	LMUONV(20)	PA identifiers of very loose muons

Usage Notes: An alternative way of calling the MUFLAG package, rather than through the MUFLAG routine itself 6.5. Once for the event, MUCALL or MUCAL2 is called and returns the number of tagged muons, and the track PA identifiers, for 3 (in the case of MUCALL) or all 4 (in the case of MUCAL2) tags from MUFLAG. Note that MUCAL2 has superceeded MUCALL with the inclusion of the VL tag.
Person Responsible: G. Wilkinson

6.7 HADSIGN

Function: Provides flags for kaon and proton identification, using the rich, the de/dx and combining them.
Source Code: Deck `HADSIGN`, Patch `HADSIGN` Cards `HADSIGN`, Library `DSTANA`, Disk `HADIDENT`.
Calling Sequence:

```
                            ===
SUBROUTINE HADSIGN(LPA,ISIGN,XK,XP,IDK,IDP,IRK,IRP)
                                  ===== == == === === === ===
```

Calling Arguments:

Input argument

`INTEGER`	`LPA`	Track pointer in the DST structure

Output arguments

`INTEGER`	`ISIGN`	Detectors used in the combination
`REAL`	`XK`	Combined flag for kaon
`REAL`	`XP`	Combined flag for proton
`INTEGER`	`IDK`	de/dx flag for kaon
`INTEGER`	`IDP`	de/dx flag for proton
`INTEGER`	`IRK`	rich flag for kaon
`INTEGER`	`IRP`	rich flag for proton

Usage Notes: All flags deal with the kaon/pion and proton/pion separation. The pure rich and de/dx flags can be equal to -1 (no information available), 0 (inconsistent), 1 (loose), 2 (standard), 3 (tight). For the combined flag, a level 0.5 has been added: 0 is for tracks identified as pions, 0.5 for tracks compatible with the pion hypothesis, and the other values have the usual meaning. The word ISIGN tells which detector has been used for the combined flag: 1 is de/dx only, 2 is rich only, 3 is both. The two rich flags (IRK,IRP) are identical to the ones returned by the richsign routine (which is called internally by hadsign). The routine reads the shortDST in a user transparent way.
Person Responsible: C. Bourdarios

6.8 RICHCER

Function: User interface to the NN algorythm, for the average cerenkov angle reconstruction. Returns for liquid and gas the 4 necessary information: average angle, sigma, number of photoelectrons and quality flag.
Source Code: Deck `RICHCER`, Patch `HADSIGN` Cards `HADSIGN`, Library `DSTANA`, Disk `HADIDENT`.
Calling Sequence:

```
                                     ===
SUBROUTINE RICHCER(LPA,THEG,SIGG,PEG,FLAGG,THEL,SIGL,PEL,FLAGL)
                                             ==== ==== === ===== ==== ==== === =====
```

Calling Arguments:

Input argument

`INTEGER`	`LPA`	Track pointer in the DST structure

Output arguments

`REAL`	`THEG`	Average angle in gas
`REAL`	`SIGG`	Sigma in gas
`REAL`	`PEG`	Number of photoelectrons in gas
`REAL`	`FLAGG`	Quality flag in gas
`REAL`	`THEL`	Average angle in liq
`REAL`	`SIGL`	Sigma in liq
`REAL`	`PEL`	Number of photoelectrons in liq
`REAL`	`FLAGL`	Quality flag in liq

Usage Notes: The quality flag MUST be used to obtain reliable results. It is coded as F1+1000*F2 , where F1 indicates if the method worked (F1.lt.100) or failed (F1.ge.100). F2 indicates if the rich was "on" for this track (F2.ge.3). This method was used for the 92D production, of nov. 93, and the routine reads the shortDST in a user transparent way. If the information is not available, or if it was not used (other shortdst processings, for example) all output words are equal to -1.
Person Responsible: C. Bourdarios for the interface routine, F. Hahn for the algorythm.

6.9 RICHWEI

Function: User interface to the WEIGHTING algorythm, for the average cerenkov angle reconstruction. Returns for liquid and gas the 4 necessary information: average angle, sigma, number of photoelectrons and quality flag.
Source Code: Deck `RICHWEI`, Patch `HADSIGN` Cards `HADSIGN`, Library `DSTANA`, Disk `HADIDENT`.
Calling Sequence:

```
                   ===
SUBROUTINE RICHWEI(LPA,THEG,SIGG,PEG,FLAGG,THEL,SIGL,PEL,FLAGL)
                       ==== ==== === ===== ==== ==== === =====
```

Calling Arguments:

Input argument

`INTEGER`	`LPA`	Track pointer in the DST structure

Output arguments

`REAL`	`THEG`	Average angle in gas
`REAL`	`SIGG`	Sigma in gas
`REAL`	`PEG`	Number of photoelectrons in gas
`REAL`	`FLAGG`	Quality flag in gas
`REAL`	`THEL`	Average angle in liq
`REAL`	`SIGL`	Sigma in liq
`REAL`	`PEL`	Number of photoelectrons in liq
`REAL`	`FLAGL`	Quality flag in liq

Usage Notes: The quality flag MUST be used to obtain reliable results. 3 levels are avialable, with an increasing purity: flag.ge.7 , 15, 31. This method will be used for the 92D and 93B production, of feb. 94, and the routine reads the shortDST in a user transparent way. If the information is not available, or if it was not used (earlier shortdst processings, for example), all output words are equal to -1.
Person Responsible: C. Bourdarios for the interface routine, P. Kluit for the algorythm.

6.10 RICHELEC

Function: Electron identification with the rich gas radiator, between .7 and 3 GeV.
Source Code: Deck `RICHELEC`, Patch `HADSIGN` Cards `HADSIGN`, Library `DSTANA`, Disk `HADIDENT`.
Calling Sequence:

```
                          ===
SUBROUTINE RICHELEC(LPA,IELEC,PROBEL)
                               ===== ======
```

Calling Arguments:

Input argument

`INTEGER`	`LPA`	Track pointer in the DST structure

Output arguments

`INTEGER`	`IELEC`	Identification flag
`REAL`	`PROBEL`	Electron probability

Usage Notes: The identification flag can be equal to -1 (no information), 0 (not consistent with an electron), 1 (loose), 2 (standard), 3 (tight).
Person Responsible: C. Bourdarios

6.11 PXRIANG

Function: For a given track crossing the rich, this routine returns information about the "environment": number of tracks crossing the same tube, on the same mirror, characteristics of the closest one.
Source Code: Deck PXRIANG, Patch HADSIGN Cards HADSIGN, Library DSTANA, Disk HADIDENT.
Calling Sequence:

```
                   ===
SUBROUTINE PXRIANG(LPA,NTUB,NMIR,DELZ,DELP,DALPHA,DTUB,NMI3)
                       ==== ==== ==== ==== ====== ==== ====
```

Calling Arguments:

Input argument

INTEGER	LPA	Track pointer in the DST structure

Output arguments

INTEGER	NTUB	number of tracks on the same tube
INTEGER	NMIR	number of tracks on the same mirror
REAL	DELZ	difference in z with the closest track
REAL	DELP	momentum of the closest track
REAL	DALPHA	angle with the closest track
REAL	DTUB	nb of tubes between this and the closest track
INTEGER	NMI3	nb of tracks above 3 GeV on the same mirror

Usage Notes: The angles are in degrees. The variable NMI3 is used in hadsign, for the identification in the vas veto region.
Person Responsible: C. Bourdarios

6.12 RICHPRBS

Function: User interface to the WEIGHTING method probabilities. For each radiator, the average angle is compared to the expectation to computes 5 probabilities (electron,mu,pi,k,proton hypotheses).
Source Code: Deck RICHPRBS, Patch HADSIGN Cards HADSIGN, Library DSTANA, Disk HADIDENT.
Calling Sequence:

```
                    ===        ======
SUBROUTINE RICHPRBS(LPA,PROBS,NPROBS,FLAGG,FLAGL)
                        =====         ===== =====
```

Calling Arguments:

Input argument

INTEGER	LPA	Track pointer in the DST structure
INTEGER	NPROBS	dimension of the PROBS array (should be 10)

Output arguments

REAL ARRAY	PROBS	10 probabilities
REAL	FLAGG	quality flag for the gas rad.
REAL	FLAGL	quality flag for the liq rad.

Usage Notes: The input argument NPROBS has to be equal to 10; remember to dimension the array probs on your code ! The flags returned are the same as for the RICHWEI routine, and MUST be used in a similar way. The probabilities are returned in the following order: liquid first (with e/mu/pi/k/p), gas second (e/mu/pi/k/p).
Person Responsible: C. Bourdarios for the interface, P.Kluit for the algorithm.

7 DEDX

Description for **VERSION 223**.
The goal of this package is to provide charged tracks identification based on the TPC dE/dx.

- BBDXID return ∼ probability of a given mass hypothesis. This routine return also information on pulls and quality flags allowing a precise use of the dE/dx information.
- BBXID same than BBDXID , but with a simplified output and some X-tra internal cut.
- BBAXID same than BBDXID , except for what concern the probability normalisation.
- BBDEDX gives access to all the parameters needed to estimate the compatibility with a given hypothesis and the measured dE/dx for a given track.
- CALDEDX same than BBDEDX but with a biased error (the measured dE/dx is used to estimate the error , instead of the expected dE/dx)
- The GETEXP/CALDEDX give access to the dE/dx expectation for a track with a given momentum and for a given mass hypothesis.
- BBDXV return all information useful for V0 reconstruction extracted using the wires and pads TPC information.
- FBBDX unpack the content of the MTPC (Xtra module =7) of the DST/SHORT DST data structure and copy it in the BBDX common . this give access to all the raw dE/dx information (only valid for DST version ¿ 333) .
- GETDXV return dedxid.car version used to read data or to produce SHORT/LONG DST.

7.1 BBDXID

Function: BBDXID, using the dE/dx measurement, returns ∼ probability of a given mass hypothesis. This ∼ probability is normalised using the mass hypothesis of the main contamination that the user want to avoid. This routine return also information on pulls and quality flags allowing a precise use of the dE/dx information.
Source Code: Deck `BBDXID`, Patch `BBXID`, Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`
.
Calling Sequence: `PROB = BBDXID(LPA,ID1,ID2,PULL1,PULL2,IQUAL)`

```
                           === === ===
      REAL FUNCTION BBDXID(LPA,ID1,ID2,PULL1,PULL2,IQUAL)
                    ======             ===== ===== =====
```

Input arguments

`INTEGER`	`LPA`	Pointer on the track in the PA bank
`INTEGER`	`ID1`	DELPHI code of the mass hypothesis you want to test
`INTEGER`	`ID2`	DELPHI code of the main contamination to the ”ID1”hypothesis.

Output argument

`REAL`	`PULL1`	Estimator for ID1 hypothesis

`REAL`	`PULL2`	Estimator for ID2 hypothesis
`INTEGER`	`IQUAL`	Quality flag = -1 No information, 0 (poor quality),.., 4(best)
`REAL`	`BBDXID`	Likelihood ratio of the 2 hypothesis

Method: The estimator computed for the "ID1" or "ID2" hypothesis is :

$$PULL = \frac{dEdx_{measured} - dEdx_{expectedforthishypothesis}}{expectederror forthishypothesis}$$

This pull is tuned to be Gaussian , centred in 0 and a sigma of 1. The likelihood ratio ($\sim$ probability) computed is

$$BBDXID = \frac{PbID1}{PbID1 + PbID2}$$

with $PbID1 = \exp(-0.5 * PULL1^2(ID1))$ and $PbID2 = \exp(-0.5 * PULL2^2(ID2))$. When there is no dE/dx information at all , BBDXID is set to -0.5 .
This BBDXID estimator provides the best ID1,ID2 separation for a given efficiency . By construction it takes into account the variation of the separation between the ID1 and ID2 hypothesis in function of the momentum and the error (=number of wires used,...).
Usage Notes: Call this routine with ID1 equal to the DELPHI code of the particle you are looking for and ID2 equal to the DELPHI code of the background you want to reject. In return you will have in BBDXID the "probability" than the current track satisfy better the hypothesis ID1 than ID2 and in PULL1 the number of sigma between the expected value for the hypothesis ID1 and the observed DEDX. In the routine a cut at 5 sigma on PULL1 is done (If $ABS(PULL1) > 5$ then BBDXID is set to 0) . The usual way to use this estimators is to cut around 0.1 - 0.2 on BBDXID and if a really pure sample is needed to cut also at 3 Sigma on PULL1.
You should be careful to not ask for a too high separation between two hypothesis if there is not enough information. Higher the value of IQUAL will be higher the cut on BBDXID can be without loosing too much in efficiency.

```
    Reminder :         DELPHI code for
                       electron --> 2
                       Muon     --> 6
                       Pion     --> 41
                       Kaon     --> 42
                       Proton   --> 65
                       Deuterium--> 39
                       Tritium  --> -39      (!)
                       Alpha    --> 40
```

Examples: K identification / Pi Veto

```
            EstiK=BBDXID(LTK,42,41,PULL1,PULL2,IQUAL)
            IF (EstiK.gt.0.2.AND.ABS(PULL1).LT.3.AND.IQUAL.GT.1) Then
*                this a Kaon ...
            ELSE IF (IQUAL.LT.1) THEN
*                not enough dE/dx information to do an identification
            END IF
```

Person Responsible: P.Antilogus

7.2 BBXID

Function: BBXID, using the dE/dx measurement, returns $\sim$ probability of a given mass hypothesis. This $\sim$ probability is normalised using the mass hypothesis of the main contamination that the user want to avoid.
Source Code: Deck `BBXID`, Patch `BBXID`, Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`
.
Calling Sequence: `PROB = BBXID(LPA,ID1,ID2)`

```
                             === === ===
      REAL FUNCTION BBXID(LPA,ID1,ID2)
                    =====
```

Input arguments

`INTEGER`	`LPA`	Pointer on the track in the PA bank
`INTEGER`	`ID1`	DELPHI code of the mass hypothesis you want to test
`INTEGER`	`ID2`	DELPHI code of the main contamination to the ”ID1”hypothesis.

Output argument

`REAL`	`BBXID`	Likelihood ratio of the 2 hypothesis

Method: The estimator computed for the ”ID1” or ”ID2” hypothesis is :

$$PULL = \frac{dEdx_{measured} - dEdx_{expectedforthishypothesis}}{expectederrorforthishypothesis}$$

then the likelihood ratio computed is

$$BBDXID = \frac{PbID1}{PbID1 + PbID2}$$

with $PbID1 = \exp(-0.5 * PULL^2(ID1))$ and $PbID2 = \exp(-0.5 * PULL^2(ID2))$. When there is not enough dE/dx information to perform a dE/dx measurement, BBXID is < 0 . This BBXID estimator provides the best ID1,ID2 separation for a given efficiency . By construction it takes into account the variation of the separation between the ID1 and ID2 hypothesis in function of the momentum and the error (=number of wires used,...).
Usage Notes: Call this routine with ID1 equal to the DELPHI code of the particle you are looking for and ID2 equal to the DELPHI code of the background you want to reject. In return you will have in BBXID the ”probability” than the current track satisfy better the hypothesis ID1 than ID2 . and in PULL1 the number of sigma between the expected value for the hypothesis ID1 and the observed DEDX. In the routine a cut at 5 sigma on PULL1 is done (If $ABS(PULL1) > 5$ then BBDXID is set to 0) . The usual way to use this estimators is to cut around 0.1 - 0.2 on BBXID.
If you want high purity sample it is better to use BBDXID with extra cut on pull and quality flag than BBXID alone.

```
Reminder :         DELPHI code for
                   electron --> 2
                   Muon     --> 6
                   Pion     --> 41
                   Kaon     --> 42
                   Proton   --> 65
                   Deuterium--> 39
                   Tritium  --> -39      (!)
                   Alpha    --> 40
```

Examples: K identification / Pi Veto

```
           EstiK=BBXID(LTK,42,41)
           IF (EstiK.gt.0.2.OR.EstiK.LT.0.) Then
*               this a Kaon ... or no dE/dx information
           END IF
```

Person Responsible: P.Antilogus

7.3 BBAXID

Function: BBAXID, using the dE/dx measurement, returns ∼ probability of a given mass hypothesis. This ∼ probability is normalised using the mass hypothesis found to be the most probable. This routine return also information on pulls and quality flags allowing a precise use of the dE/dx information.
Source Code: Deck `BBAXID`, Patch `BBXID`, Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`
.
Calling Sequence: `PROB = BBAXID(LPA,ID1,ID2,PULL1,PULL2,IQUAL)`

```
                            === ===
      REAL FUNCTION BBAXID(LPA,ID1,ID2,PULL1,PULL2,IQUAL)
                    ======         === ===== ===== =====
```

Input arguments		
`INTEGER`	`LPA`	Pointer on the track in the PA bank
`INTEGER`	`ID1`	DELPHI code of the mass hypothesis you want to test
Output argument		
`INTEGER`	`ID2`	DELPHI code of the most probable hypothesis.
`REAL`	`PULL1`	Estimator for ID1 hypothesis
`REAL`	`PULL2`	Estimator for ID2 hypothesis
`INTEGER`	`IQUAL`	Quality flag = -1 No information, 0 (poor quality),.., 4(best)
`REAL`	`BBDXID`	Likelihood ratio of the 2 hypothesis

Method: The estimator computed for the "ID1" or "ID2" hypothesis is :

$$PULL = \frac{dEdx_{measured} - dEdx_{expectedforthishypothesis}}{expectederrorforthishypothesis}$$

then the likelihood ratio computed is

$$BBAXID = \frac{PbID1}{PbID1 + PbID2}$$

with $PbID1 = \exp(-0.5 * PULL1^2(ID1))$ and $PbID2 = \exp(-0.5 * PULL2^2(ID2))$. When there is no dE/dx information at all , BBAXID is set to -0.5 .

This BBDXID estimator provides the best ID1,ID2 separation for a given efficiency . By construction it takes into account the variation of the separation between the ID1 and ID2 hypothesis in function of the momentum and the error (=number of wires used,...).

Usage Notes: Call this routine with ID1 equal to the DELPHI code of the particle you are looking for. In return you will have in ID2 the DELPHI code of the mass hypothesis with the PULL value closest to 0 (excluding ID1) and in BBDXID the "probability" than the current track satisfy better the hypothesis ID1 than ID2. In PULL1 (or PULL2) the number of sigma between the expected value for the hypothesis ID1 (or ID2) and the observed DEDX. In the routine a cut at 5 sigma on PULL1 is done (If $ABS(PULL1) > 5$ then BBDXID is set to 0) . The usual way to use this estimators is to cut around 0.1 - 0.2 on BBAXID and if a really pure sample is needed to cut also on PULL1 or PULL2.

You should be careful to not ask for a too high separation between two hypothesis if there is not enough information. Higher the value of IQUAL will be higher the cut on BBAXID can be without loosing too much in efficiency.

Certainly the best use of the output of this routine will be as a cross check of the BBDXID selection , namely to be sure that there is no contamination from other sources than the one considered in BB-DXID.

```
        Possible  ID1 value           checked  ID2 value :
           electron --> 2                  Pi,K,p
           Muon     --> 6                  e,k,p
           Pion     --> 41                 e,k,p
           Kaon     --> 42                 e,pi,p
           Proton   --> 65                 e,pi,k
           Deuterium--> 39                 e,pi,k,p,tr,alpha
           Tritium  --> -39                e,pi,k,p,de,alpha
           Alpha    --> 40                 e,pi,k,p,tr,de
                      ID1 can also take the heavy tag value :
           heavy    --> 107 (=42+65=p or k)e,pi
```

Examples: Heavy track identification identification

```
            Estih=BBAXID(LTK,107,ID2,PULL1,PULL2,IQUAL)
            IF (Estih.gt.0.8.AND.ABS(PULL1).LT.3.AND.IQUAL.GT.2) Then
*                this a Kaon or a proton ...
            ELSE IF (IQUAL.LT.1) THEN
*                not enough dE/dx information to do an identification
            END IF
```

Person Responsible: P.Antilogus

7.4 BBDEDX

Function: BBDEDX returns the measured dE/dx , the expectation/error for a given mass hypothesis and some information on the quality of the dE/dx measurement.
Source Code: Deck `BBDEDX`, Patch `BBXID`, Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`
.
Calling Sequence: `CALL BBDEDX(LPA,ID1,DEDX,NWIR,DEDXH,ERROR,IQUAL)`

```
                      === ===
    SUBROUTINE BBDEDX(LPA,ID1,DEDX,NWIR,DEDXH,ERROR,IQUAL)
                              ==== ==== ===== ===== =====
```

Input arguments

`INTEGER`	`LPA`	Pointer on the track in the PA bank
`INTEGER`	`ID1`	DELPHI code of the mass hypothesis you want to test

Output argument

`REAL`	`DEDX`	Measured dE/dx (80%) normalized to 1 for MIP.
`INTEGER`	`NWIR`	Number of wires used in the measurement.
`REAL`	`DEDXH`	expected dE/dx for mass hypothesis ID1
`REAL`	`ERROR`	expected error for the measured dE/dx and the mass hypothesis ID1.
`INTEGER`	`IQUAL`	Quality flag = -1 No information, 0 (poor quality),.., 4(best)

Usage Notes: Call this routine with ID1 equal to the DELPHI code of the particle you are looking for. The estimation of the error done by this routine is much more reliable than in GETDEDX. A cut on the number of wires is not enough to garanty a good dE/dx measurement , the quality flag IQUAL is more reliable as it takes into account the length of the track measured in the TPC. It should also be noticed that the pull wich can be computed with DEDX , DEDXH and ERROR is Gaussian. This is due to non symmetric fluctuation of the dE/dx. To access a Gaussian pull the function BBDXID or BBAXID should be used.

```
    Reminder :         DELPHI code for
                       electron --> 2
                       Muon     --> 6
                       Pion     --> 41
                       Kaon     --> 42
                       Proton   --> 65
                       Deuterium--> 39
                       Tritium  --> -39      (!)
                       Alpha    --> 40
```

Person Responsible: P.Antilogus

7.5 GETDEDX

Function: Gives the dE/dx value from the DST for a PA bank.

Source Code: Deck `GETDEDX`, Patch `DEDXID` Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`
.
Calling Sequence:CALL GETDEDX(LPA,DEDX,SIGM,NWIR,ERROR)

```
                   ===
SUBROUTINE GETDEDX(LPA,DEDX,SIGM,NWIR,ERROR)
                       ==== ==== ==== =====
```

Calling Arguments:

Input argument

`INTEGER`	`LPA`	Track pointer in the DST structure

Output arguments

`REAL`	`DEDX`	measured dE/dx (80%)
`REAL`	`SIGM`	Sigma of the measured Landau distribution
`INTEGER`	`NWIR`	80% of the number of wires with dE/dx information
`REAL`	`ERROR`	estimated error on dE/dx .

Usage Notes: It is not recommended to use this routine any more. BBDXID or BBDEDX should be used instead. For old DELANA/DST version (before PXDST 333) , the use of one or the other routine will not make a difference. But for more recent processing the BBDEDX code should provide more reliable error calculation and BBDXID better pull. **Person Responsible:** P.Antilogus

7.6 CALDEDX

Function: Calculates the expected dE/dx value for a given mass and momentum.
Source Code: Deck `CALDEDX`, Patch `DEDXID` Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`.
Calling Sequence:`CALL CALDEDX(PMOM,RMASS,DEDX,DSURDP)`

```
                         ==== =====
       SUBROUTINE CALDEDX(PMOM,RMASS,DEDX,DSURDP)
                                    ==== ======
```

Calling Arguments:

Input arguments

`REAL`	`PMOM`	Momentum value
`REAL`	`RMASS`	Mass value

Output arguments

`REAL`	`DEDX`	Value of dE/dx
`REAL`	`DSURDP`	Derivative of dE/dx with momentum

Usage Notes: The routine GETEXP is may be of an easier usage .
Person Responsible: P.Antilogus

7.7 GETEXP

Function: Calculates the expected dE/dx value for a given particle hypothesis and momentum.
Source Code: Deck `GETEXP`, Patch `DEDXID` Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`.
Calling Sequence:`CALL GETEXP(P,ID1,DEDX,DSURDP)`

```
                   ==== =====
      SUBROUTINE GETEXP(P,M,DEDX,DSURDP)
                              ==== ======
```

Calling Arguments:

Input arguments

`REAL`	`PMOM`	Momentum value
`INTEGER`	`ID1`	DELPHI code of the particle

Output arguments

`REAL`	`DEDX`	Value of dE/dx
`REAL`	`DSURDP`	Derivative of dE/dx with momentum

Usage Notes:

```
    Reminder :         DELPHI code for
                       electron --> 2
                       Muon     --> 6
                       Pion     --> 41
                       Kaon     --> 42
                       Proton   --> 65
                       Deuterium--> 39
                       Tritium  --> -39      (!)
                       Alpha    --> 40
```

Example: Expected dE/dx for electron

```
            LMAIN=LPHPA(LPA,'MAIN',0)
            CALL GETEXP(Q(LMAIN+7),2,DEDX,DSURDP)
```

Person Responsible: P.Antilogus

7.8 BBDXV

Function: Return useful information for V0 reconstruction. Precise estimation of the position in RPHI of a decay vertex in the TPC is extracted from the TPC wires information.
Source Code: Deck `BBDXV`, Patch `DEDXID` Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`.
Calling Sequence:`OK = BBDXV(LPA,IWF,ISF,YF,PHIF,IBV0,IPF)`

```
                            ===
      LOGICAL FUNCTION BBDXV(LPA,IWF,ISF,YF,PHIF,IBV0,IPF)
                                 === === == ==== ==== ===
```

Calling Arguments:

Input arguments

`INTEGER`	`LPA`	Pointer on the track in the PA bank

Output arguments

`INTEGER`	`IWF`	first wire seen by the track
`INTEGER`	`ISF`	first sector seen by the track
`REAL`	`YF`	”yf” corresponding y coordinate in the frame of the sector
`REAL`	`PHIF`	the sector is rotated by ”phif” compared to the DELPHI frame.
`INTEGER`	`IBV0`	bit pattern of the TPC pad row used in the track * fit (bit 1-¿16)
`INTEGER`	`IPF`	If different from 0 it gives the first pad row where this track has been seen.
`LOGICAL`	`BBDXV`	= .TRUE. if TPC information available

Usage Notes: Should be used one day to improve the position in the TPC of vertices from V0 decays . The distance between 2 wires in the TPC is 4 mm , allowing a resolution of 1.2 mm on the position of the vertex in RPHI. The use of this information should reduce the mass shift due to the bad estimated position vertex in the TPC for track close the one to the other.

Person Responsible: P.Antilogus

7.9 FBBDX

Function: This routine can be used to unpack the MTPC X-tra module (=7) and to put it in the BBDX common .
Source Code: Deck `FBBDX`, Patch `BBXID` Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`.
Calling Sequence:`OK = FBBDX(LPA,NDEDX)`

```
                             === =====
      LOGICAL FUNCTION FBBDX(LPA,NDEDX)
```

Calling Arguments:

Input arguments

`INTEGER`	`LPA`	>0 pointer on a track in the PA bank, the BBDX common will be filled =0 Reset the BBDX common and fill the pressure and other general event information in BBDX (should be done once per event if you need such information) <0 Then abs(ltk) is a ntuple identifier and a HBNAME declaration will be done between this ntuple and the BBDX common
`INTEGER`	`NDEDX`	Index to be used when filling the BBDX common (can be always 1 if you want)

Output arguments

`LOGICAL`	`BBDXV`	= .TRUE. if dE/dx information available for this track.

Usage Notes: After a call to this routine the BBDX common contains :

```
* ======= Index
*      NBDX               Last value of NDEDX
* ======= general event information
*      PRESS              Pressure
*      XNORM              Normalisation used for the MIP
```

```
*      RATIO              Ratio between INT and MAX amplitude used
* ======= Main DEDX information for isolated track
*      DEDX80(NDEDX)      DEDX Max amplitude 80%    ( The usual DEDX )
*      SIGM80(NDEDX)      Sigma of the Truncated 80% LANDAU (not an "error" !)
*      NWTOT(NDEDX)       Total number of wire before truncation
*                         ( number of wire used for DEDX80 = NINT(.8*NTOT) )
*      DEDX60(NDEDX)      DEDX MAx amplitude 60%
*      SIGM60(NDEDX)      Corresponding Sigma
*      DEDX80I(NDEDX)     DEDX Int amplitude 80%
*      DXL                DEDX Max amplitude Full Landau
*      DXLM2,DXLM3        Moment of order 2 and 3 of the Landau
* ======= Information  on the DEDX, can be used for error or quality estimation
*      NWCRO(NDEDX)       Nb wires crossed by the track (all may not be used )
*      NWCLC(NDEDX)       Nb of wire with a close cluster in Z
*      DZCLS(NDEDX)       Average distance in Z to these clusters
*      NMETH(NDEDX)       = 1 Good track, enough wires away from the edge
*                         = 0 Lots of wire at the edge of TPC/sectors
*      NWEDG(NDEDX)       Number of hits at the edge of the TPC/sectors
*                         ( if NMETH = 1 they are included in the average )
*      EDGE(NDEDX)        Average distance to the edge of the TPC sectors
*      NWSAT(NDEDX)       Number of saturated hits
*      NWEMPT(NDEDX)      Number of empty wires ( hits below threshold )
*      DEDX0(NDEDX)       Estimated Weight of the empty hits in DEDX80
*      UNSH(NDEDX)        Estimated weight of the Undershoot correction
*                         in DEDX80
*      GAP(NDEDX)         Average GAP size
* ======== Information for overlapping tracks/hits
*      DX280(NDEDX)       DEDX Int amplitude truncated at 80%
*      SI280(NDEDX)       Sigma of the DEDX
*      N2WIRE(NDEDX)      Number of common  wires used in the DEDX estimation
*                         ( Number of wire used for DX280 = NINT(0.8*N2WIRE)
*      N2WSAT(NDEDX)      Number of saturated hits( can be equal to 2*N2wire )
*      GAP2(NDEDX)        Average Gap
*      DX2L(NDEDX)        DEDX of the Full Landau
*      SI2L(NDEDX)        Sigma of the Full Landau
*      N2IDX(NDEDX)       Internal TPCANA Index of the current track
*  ABS(N2ASS(NDEDX))      Internal TPCANA Index of the overlapping track
*                         (N2ASS sign is related to unpacking info....)
*      LCUR(NDEDX)        PA pointer on the current track (protected)
*      LASS(NDEDX)        PA pointer on the associated/overlapping track (//)
* ======= Information on the fit of the track ( PAD)
*      NPAD(NDEDX)        Number of PAD used in the TPC TE
*      NSIN(NDEDX)        First sector seen by the track/TE in the TPC
*      NSOU(NDEDX)        Last sector seen by the track/TE in the TPC
* ======== Information on the Fit in Z
*      NWZ(NDEDX)         Number of wires used in the fit in Z
*      ZCHI2(NDEDX)       Chi2 of the fit in Z
```

```
* ======== Information for V0 reconstruction
*      NPHIT(NDEDX)       Bit pattern of the pad with a hit for this track
*      NPV0(NDEDX)        = 0 not V0 flag , 1-16 First Raw seen for the V0
*      NWFS(NDEDX)        First Wire with a hit from this track
*                        (the hit may not be used in the dE/dx or Z calculation)
*      NWFW(NDEDX)        Corresponding First sector
*      NWLW(NDEDX)        Last Wire with a hit from this track
*      NWLS(NDEDX)        Corresponding Last sector
* ==========================================================================
```

Example: Save information in ntuple for all tracks with TPC info.

```
* CDE
+CDE,PHQ.
+CDE,BBDX.
* Local var
            LOGICAL FIRST
            SAVE    FIRST
            DATA    FIRST/.TRUE./
* initialisation
            IF (FIRST) THEN
              CALL HBNT(1,'dE/dx for all track',' ')
              FIRST=FBBDX(-1,0)
              FIRST=.FALSE.
            END IF
* Loop on track
            NUM=1
            LPV=LQ(LDTOP-1)
            DO WHILE (LPV.NE.0)
              LPA=LQ(LPV-1)
              DO WHILE (LPA.NE.0)
* Look for TPC information
                IF (FBBDX(LPA,NUM))  NUM=NUM+1
                LPA=LQ(LPA)
              END DO
              LPV=LQ(LPV)
            END DO
* save the ntuple if needed
            IF (NUM.GT.1) then
* Need to overwrite NBDX , has the last track could be not selected.
              NBDX=NUM-1
              CALL HFNT(1)
            END IF
```

Person Responsible: P.Antilogus

7.10 GETDXV

Function: Gives access to the version used of dedxid.car ,including the version used for SHORT or LONG DST production (if dedxid version older than 222).

Source Code: Deck `GETDXV`, Patch `DEDXID` Cards `dedxid.car`, Library `DSTANA`, Disk `$DELPHI_PAM`.
Calling Sequence:`CALL GETDXV(ITYPE,IV)`

```
                            =====
       SUBROUTINE GETDXV(ITYPE,IV)
                                  ==
```

Calling Arguments:

Input arguments

`INTEGER`	`ITYPE`	From where the dedxid version should be taken ? if = 0 it is taken from the dedxid.car pam file and if = 1 is taken from the SHORT/LONG pilot (not implemented yet)

Output arguments

`INTEGER`	`IV`	dedxid/bbxid version number

Person Responsible: P.Antilogus

7.11 RECV0

Function: Finds all V0s in an event and stores useful parameters in a common block
Source Code: Deck UUSFL, Patch RECV0, Cards RECV0, Library DSTANAXX, Disk DSTANA.
Calling Sequence:

```
                          ===== ====== ==== ==== ====
SUBROUTINE RECV0(XYZBS,SIGXYZ,CUT1,CUT2,CUT3)
```

Calling Arguments:

Input arguments

REAL	XYZBS(3)	Co-ordinates of beam spot in 3 dimensions
REAL	SIGXYZ(3)	Errors on XYZBS(3)
REAL	CUT1	Lower limit for chi-squared probability from V0 vertex fit
REAL	CUT2	Lower limit for scaled distance in r-phi plane between beam spot and V0 vertex
REAL	CUT3	Upper limit for angle(in radians) between reconstructed V0 momentum and vector joining beam spot and V0 vertex (in r-phi)

Output arguments (contained in common block NSTAL1)

INTEGER	NST1	Number of V0s found satisfying above cuts
INTEGER	IPNL1(NST1,2)	Links to PAs of V0 daughters
REAL	PNL1(NST1)	Magnitude of V0 momentum
REAL	PXY1(NST1)	Magnitude of V0 momentum in r-phi
REAL	PCHI1(NST1)	Chi-squared probability of V0 vertex fit
REAL	VTXS1(NST1,3)	Position of V0 vertex
REAL	DXY1(NST1)	Scaled distance(with respect to a combination of beam spot and secondary vertex errors) in r-phi of V0 vertex from beam spot
REAL	ANGLE1(NST1)	angle(in radians) between reconstructed V0 momentum and vector joining beam spot and V0 vertex (in r-phi plane)
REAL	DZ1(NST1)	separation in z of the V0 daughters at the fitted vertex
REAL	PPOS1(NST1)	momentum of positively charged V0 daughter
REAL	PNEG1(NST1)	momentum of negatively charged V0 daughter
REAL	COTH1(NST1)	angle betweeen V0 daughters in r-phi plane
REAL	PPAR1(NST1,3,2)	momenta of V0 daughters at fitted vertex
REAL	VMNL1(NST1)	reconstructed mass for L0 hypothesis
REAL	VMNPH1(NST1)	reconstructed mass for photon hypothesis
REAL	VMNK01(NST1)	reconstructed mass for K0 hypothesis
REAL	RJ(NST1)	ratio of DXY1 and DZ1 multiplied by a factor which depends on the number of rightly/wrongly associated VD hits on the V0 daughters

Usage Notes: V0 selextion uses the following cuts in addition to those specified by the user: a) the number of VD hits associated to V0 daughters must be consistent with the position of the decay vertex; b) daughters must be associated to one and only one V0; c) the transverse momentum of each daughter must be greater than 20MeV with respect to the sum.
Person Responsible: M. Caccia, E. Torassa, A. deAngelis

7.12 MASSV0

Function: Re-calculates the vertex co-ordinates and track parameters of a previously fitted 2-prong vertex, according to a mass constraint. **Source Code:** Deck MASSV0, Patch MASSV0, Cards MASSV0, Library DSTANAXX, Disk DSTANA.
Calling Sequence:

```
                   ===== ===== ==== === ==== == == ==== ==== =====
SUBROUTINE MASSV0(AMAS1,AMAS2,AMAS,XYZ,VCOV,I1,I2,PARF,TCOV,CONSB,
                   XYZC,PARC,VCOVC,TCOVC,TTCOVC)
                   ==== ==== ===== ===== ======
```

Calling Arguments:

Input arguments

REAL	AMAS1	Mass of first track
REAL	AMAS2	Mass of second track
REAL	AMAS	Constraining mass for combination
REAL	XYZ(3)	Co-ordinates returned by vertex fit
REAL	VCOV(6)	Triangular covaraince matrix of XYZ
INTEGER	I1,I2	Indicies (1,2) of the two tracks
REAL	PARF(3,2)	Track parameters(theta,phi,1/R) at XYZ
REAL	TCOV(6,2)	Triangular covaraince matrix of PARF
REAL	CONSB	Magnetic field in units of GeV/c/cm

Output arguments

REAL	XYZC(3)	Mass-constrained co-ordinates of vertex
REAL	VCOVC(6)	Triangular covaraince matrix of XYZC
REAL	PARC(3,2)	Mass-constrained track parameters at XYZC
REAL	TCOVC(6,2)	Triangular covaraince matrix of PARC
REAL	TTCOVC(6,2)	Covariance matrix of track-track correlations

Usage Notes: PXFVTX or some similar secondary vertex fitting routine must be called previously to obtain the input parameters.
Person Responsible: A. DeAngelis

7.13 MASUSE

Function: Re-calculates the vertex co-ordinates and track momenta of a previously fitted 2-prong vertex, when the invaraint mass of the vertex is constrained to be that of a L0/K0. **Source Code:** Deck `MASUSE`, Patch `MASSV0`, Cards `MASSV0`, Library `DSTANAXX`, Disk `DSTANA`.
Calling Sequence:

```
                  ==== ==== ====
SUBROUTINE MASUSE(LPA1,LPA2,VTXL,QFV0,VTXF,IERR)
                                 ==== ==== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`LPA1,LPA2`	PA links of tracks
`REAL`	`VTXL(3)`	Co-ordinates returned by vertex fit

Output arguments

`REAL`	`QFV0(4,1)`	Constrained 4-momentum of V0 for K0 mass
`REAL`	`QFV0(4,2)`	Constrained 4-momentum of V0 for L0 mass
`REAL`	`VTXF(3,1)`	Constrained co-ordinates of vertex for K0 mass
`REAL`	`VTXF(3,1)`	Constrained co-ordinates of vertex for L0 mass
`INTEGER`	`IERR`	Error flag; 0=ok

Usage Notes: `PXFVTX` or some similar secondary vertex fitting routine must be called previously to obtain the input parameters.
Person Responsible: A. DeAngelis

8 Vertex Physics Routines

These routines are to manipulate the vertex information in the events. They are independent of `VDCLAP` (section 11) in so far as they read the information directly from the DST.

8.1 READBS

Function: Reads in the run-by-run beamspot files.
Source Code: Deck READBS, Patch GRACOM, Cards GRACOM, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                    ===
SUBROUTINE READBS(LUN)
```

Calling Arguments:

Input argument

INTEGER	LUN	Logical unit number for beamspot file

Usage Notes: The beamspot can be manipulated by MOVEBS or VDBSOF (section 8.2) and SETBS (section 8.3). The run-by-run beamspot is given by VDBEAM or VDBMCA (section 8.4). The beam spot file must have been assigned to the logical unit before running.
Note that several beamspot files can be read in sequentially for different years. If the *same* run number is read twice the results are not defined.
Person Responsible: W. Murray

8.2 MOVEBS, VDBSOF

Function: Gives an overall movement in x, y, z of beamspot reported by `VDBEAM`.
Source Code: Deck `MOVEBS`, Patch `GRACOM`, Cards `GRACOM`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                  =====
SUBROUTINE MOVEBS(BSXYZ)

                  =====
SUBROUTINE VDBSOF(BSXYZ)
```

Calling Arguments:

Input argument

`REAL`	`BSXYZ(3)`	Beamspot shift in x, y, z

Usage Notes: This routine enables misalignments between the current data and the beamspot file to be corrected. The beamspots must have previously been read in by `READBS` (section 8.1). For `MOVEBS`, the actual stored value of each beamspot is changed by the routine, so if it is called twice the effects are additive. For `VDBSOF`, the shift is stored separately, so the shift is purely that specified by the last call.
Person Responsible: W. Murray

8.3 SETBS

Function: Sets definition of error or size reported by `VDBEAM` (section 8.4).
Source Code: Deck `SETBS`, Patch `GRACOM`, Cards `GRACOM`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                  ====== ====== ======
SUBROUTINE SETBS(IFOLDI,IMCBSI,SIGMAI)
```

Calling Arguments:

Input arguments

`INTEGER`	`IFOLDI`	Error option; 0 (default) = error returned by `VDBEAM` is error on average only, 1 = error returned is error on average quadratically added with the beamspot size
`INTEGER`	`IMCBSI`	Simulation option; 0 (default) = beamspot returned is average of simulation events 1 = gaussian distribution around top PV bank.
`REAL`	`SIGMAI(3)`	Beamspot size and/or gaussian distribution size

Usage Notes: The beamspot file must have previously been read in by `READBS` (section 8.1).
The simulation option `IMCBSI=1` effectively allows any chosen beamspot size to be used with simulation events, independent of what beamspot size was used during the generation. The position defined as the "average" is randomly positioned around the actual production point for that event, so that the separation can mimic any required size of beamspot.
Person Responsible: W. Murray

8.4 VDBEAM, VDBMCA

Function: Gives the average beamspot per event
Source Code: Deck VDBEAM, Patch GRACOM, Cards GRACOM, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                  ==== ====
SUBROUTINE VDBEAM(NRUN,IEVT,XYZV,DXYZV,IERR)
                            ==== ===== ====

                  ==== ======
SUBROUTINE VDBMCA(NRUN,NFILSQ,XYZV,DXYZV,IERR)
                              ==== ===== ====
```

Calling Arguments:

Input arguments

INTEGER	NRUN	Run number
INTEGER	IEVT	Event number
INTEGER	NFILSQ	Online file number

Output arguments

REAL	XYZV(3)	Position of average beamspot
REAL	DXYZV(3)	Error or size of average beamspot
INTEGER	IERR	Error flag; 0 = fine, 1 = marginal, 2 = unusable

Usage Notes: The beamspot file must have previously been read in by READBS (section 8.1).
VDBEAM is obselete; if NFILSQ is speficied in the /UXCEVT/ common, it simply calls VDBMCA. If NFILSQ is zero, it attempts to calculate a reasonable value from IEVT. However, it is better to call VDBMCA directly.
The switching between DXYZV being the size or the error is done by SETBS (section 8.3).
Person Responsible: W. Murray

8.5 DPAGIP

Function: Finds the geometric-signed impact parameter for a track relative to a beam position.
Source Code: Deck DPAGIP, Patch DSTIO, Cards DSTANA, Library DSTANA, Disk PHY.
Calling Sequence:

```
                    ====== ======
SUBROUTINE DPAGIP(LDSTPA,BEAMXY,BGSIP,EGSIP)
                                  ===== =====
```

Calling Arguments:

Input arguments

INTEGER	LDSTPA	Link to PA bank
REAL	BEAMXY(2)	Beamspot x,y

Output arguments

REAL	BGSIP	Geometric-signed impact parameter
REAL	EGSIP	Error on the impact parameter

Usage Notes: The impact parameter is defined in the $R\Phi$ plane only.
In principle, the perigee parameter errors on some of the DST productions (before 92D) are incorrect, as all multiple scattering to the perigee is not included. To get the "correct" errors, first call DSTMSP (section 10.12) or DPAMSP (section 10.13).
Person Responsible: P. Dauncey

8.6 DPALIP

Function: Finds the lifetime-signed impact parameter for a track relative to a beam position, given a jet direction.
Source Code: Deck `DPALIP`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                  ====== ====== ====
SUBROUTINE DPALIP(LDSTPA,BEAMXY,PJET,BLSIP,ELSIP)
                                     ===== =====
```

Calling Arguments:

Input arguments

`INTEGER`	`LDSTPA`	Link to PA bank
`REAL`	`BEAMXY(2)`	Beamspot x,y
`REAL`	`PJET(2)`	Jet x,y components (need not be normalised)

Output arguments

`REAL`	`BLSIP`	Lifetime-signed impact parameter
`REAL`	`ELSIP`	Error on the impact parameter

Usage Notes: The impact parameter is defined in the $R\Phi$ plane only.
In principle, the perigee parameter errors on some of the DST productions (before 92D) are incorrect, as all multiple scattering to the perigee is not included. To get the "correct" errors, first call `DSTMSP` (section 10.12) or `DPAMSP` (section 10.13).
Person Responsible: P. Dauncey

8.7 DPAIP3

Function: Computes the life-time signed impact parameter in space, its error, the minimal track-jet distance end its error.
Source Code: Deck DPAIP3, Patch BTAGGING, Cards DSTANA, Library DSTANA, Disk PHY.
Calling Sequence:

```
                  === == === == ===
SUBROUTINE DPAIP3(LPA,VP,SVP,JD,SJD,DJ,SDJ,IP3,SIP3,IERR)
                                    == === === ==== ====
```

Calling Arguments:

Input arguments

INTEGER	LPA	Link to the PA bank
REAL	VP(3)	Primary vertex position
REAL	SVP(6)	Primary vertex position covariance matrix
REAL	JD(3)	Jet momentum
REAL	SJD(6)	Jet momentum covariance matrix

Output arguments

REAL	DJ	Minimal track-jet distance
REAL	SDJ	Error on the inimal track-jet distance
REAL	IP3	Life time signed impact parameter in space
REAL	SIP3	Error on the life time signed impact parameter in space
INTEGER	IERR	Error flag

Usage Notes: This routine computes the impact parameter in space and the minimal track-jet distance and their errors. The errors are computed with a linear propagation from the input covariance matrices, taking into account the correlations. The sign on the impact parameter is life-time dependent.
Person Responsible: P. Rebecchi

8.8 DSTVPX

Function: Calculates a vertex for a given list of PA banks in the $R\Phi$ plane or in $R\Phi z$.
Source Code: Deck `DSTVPX`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                        ==== ====== ======
SUBROUTINE DSTVPX(NDIM,NDSTPA,LDSTPA,VERTEX,VERROR,CHISQ,NDOF,CHISQT)
                                         ====== ====== ===== ==== ======
```

Calling Arguments:

Input arguments

`INTEGER`	`NDIM`	Number of dimensions; 2 = $R\Phi$ only, 3 = $R\Phi z$
`INTEGER`	`NDSTPA`	Number of PA banks to be used
`INTEGER`	`LDSTPA(*)`	List of links to PA banks

Output arguments

`REAL`	`VERTEX(2 or 3)`	Fitted vertex position in x,y or x,y,z
`REAL`	`VERROR(3 or 6)`	Fitted vertex error matrix (triangle)
`REAL`	`CHISQ`	Chi-squared of fit
`INTEGER`	`NDOF`	Number of degrees of freedom of fit
`REAL`	`CHISQT(*)`	Chi-squared of each track; negative if not used

Usage Notes: This routine is an interface to `PXFVTX` (section 8.9).
In principle, the perigee parameter errors on some of the DST productions (before 92D) are incorrect, as all multiple scattering to the perigee is not included. To get the “correct” errors, first call `DSTMSP` (section 10.12).
Person Responsible: P. Dauncey

8.9 PXFVTX, VDF2ND

Function: Calculate a vertex from a set of perigee parameters.
Source Code: Deck PXFVTX, Patch PXFVRT, Cards PXDST, Library PXDST, Disk PHY; and Deck VDF2ND, Patch REC2ND, Cards REC2ND, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                              ==== ===== ===== === === ===
SUBROUTINE PXFVTX(       NTRK,FITWB,INVTX,PAR,WGT,XYZ,
                 XYZF,PARF,VCOV,TCOV,CHI2,CHI2TR,IERR)
                 ==== ==== ==== ==== ==== ====== ====

                 ====== ==== ===== ===== === === ===
SUBROUTINE VDF2ND(MXITER,NTRK,FITWB,INVTX,PAR,WGT,XYZ,
                 XYZF,PARF,VCOV,TCOV,CHI2,CHI2TR,IERR)
                 ==== ==== ==== ==== ==== ====== ====
```

Calling Arguments:

Input arguments

INTEGER	MXITER	Maximum iterations over PXFVTX
INTEGER	NTRK	Number of set of parameters to be used
LOGICAL	FITWB	Use beam constraint or not
LOGICAL	INVTX(*)	Use this track in the fit or not
REAL	PAR(5,*)	Perigee parameters
REAL	WGT(15,*)	Perigee parameter weight matrices
REAL	XYZ(3)	Initial guess for vertex

Output arguments

REAL	XYZF(3)	Fitted vertex
REAL	PARF(3,*)	Fitted parameters θ, ϕ, κ
REAL	VCOV(6)	Error matrix on the vertex
REAL	TCOV(6,*)	Error matrix on the parameters
REAL	CHI2	χ^2 of fit
REAL	CHI2TR(*)	χ^2 of each track
INTEGER	IERR	Error flag; 0 is OK

Usage Notes: This vertex is calculated using a full three-dimensional fit and allows all 5 track parameters to vary. VDF2ND simply iterates over PXFVTX, allowing it to be used when the fitted vertex is far from the initial guess.
Person Responsible: P. Billoir (PXFVTX), T. Adye (VDF2ND).

8.10 REC2ND, VDMPER

Function: Reconstruct the parent's mass and 4-momentum, with their errors, from the decay products' track parameters and errors as returned by PXFVTX/VDF2ND (section 8.9).
Source Code: Decks REC2ND and VDMPER, Patch REC2ND, Cards REC2ND, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
+CDE, PUCPPP.
                  ==== ====== ==== ==== ===== ==
SUBROUTINE REC2ND(NTRK,USECOR,PARF,TCOV,PMASS,IV,D4P)
                                                 ===

                  == ===
SUBROUTINE VDMPER(PE,D4P,DP,DM,DPHI)
                         == == ====
```

Calling Arguments:

Input arguments

INTEGER	NTRK	Number of tracks. NB. All are included here.
INTEGER	USECOR	Use track–track correlations? 0=no, 1=yes
REAL	PARF(3,*)	Track parameters at decay point θ, ϕ, κ
REAL	TCOV(6,*)	Error matrix on the parameters
REAL	PMASS(*)	Assumed daughter particle masses
INTEGER	IV	Output momenta in VECP(*,IV:IV+NTRK)
REAL	PE(4)	4-momentum: specify as VECP(1,IV) to use REC2ND output

Output arguments

REAL	D4P(10)	Error matrix on reconstructed 4-momentum
REAL	DP	Error on the momentum
REAL	DM	Error on the mass
REAL	DPHI	Error on the ϕ direction

Usage Notes: REC2ND combines track direction and curvature parameters and errors returned by PXFVTX/VDF2ND (section 8.9) to reconstruct the parent's 4-momentum (returned in VECSUB's VECP(*,IV)) and errors (returned packed into D4P). The daughter tracks' 4-momenta are returned in VECP(*,IV+1:IV+NTRK). If USECOR=1, then track–track correlations must be included in TCOV(6,NTRK,NTRK) (see cards file for details on how this may be extracted from VDF2ND).
VDMPER transforms a 4-momentum (which can be taken from VECP(*,IV), as returned by REC2ND) and its error matrix (D4P) into errors on the momentum, mass, and ϕ direction.
Person Responsible: T. Adye

8.11 CALPER

Function: Calculate the perigee parameters and their errors for a new ”pseudo-track”.
This routine can be called after a secondary vertex fit has been performed, see section 8.9. It creates a pseudo-track of which the perigee parameters and errors can easily be used in following vertex fits. The original code can be found in the `SUBROUTINE TBFITS` of `CORPUS`.
Source Code: Deck `CALPER`, Patch `REC2ND`, Cards `REC2ND`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                          == ===== ==== ===== ===== =====
      SUBROUTINE CALPER(NT,VXNEW,PARF,VXCOV,TKCOV,IQVEE,
     +                  QVAT,QVOF,PPER,EPER)
                          ==== ==== ==== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`NT`	Number of tracks belonging to the vertex
`REAL`	`VXNEW(3)`	Coordinates of the fitted vertex
`REAL`	`PARF(3,NT)`	Parameters of the tracks at the fitted vertex
`REAL`	`VXCOV(6)`	Vertex error matrix
`REAL`	`TKCOV(6,NT)`	Error matrix on the θ, ϕ, $1/R$ fitted parameters for each track
`INTEGER`	`IQVEE`	Total charge of the participating tracks

Output arguments

`REAL`	`QVAT(10,NT)`	Parameters of the tracks at the secondary vertex, filled according to the `VECSUB` convention
`REAL`	`QVOF(10)`	Total vertex parameters
`REAL`	`PPER(5)`	Parameters of the pseudo-track originating from the vertex
`REAL`	`EPER(15)`	Error matrix for the perigee parameters. It must converted into the weight matrix if it is to be used in second vertex fit. This can be easily done by a call to `FKMI5`.

Usage Notes:
This is an example of the recommended calling sequence :

```
      CALL VDF2ND(MXITER,NT,FITWB,INVTX,PAR,WGT,XYZ,VERT,PARF,VERR,
                  TCOV,CHISQ,DCHIST,IERR)
      CALL CALPER(NT,VERT,PARF,VERR,TCOV,ICHGTOT,
                  QVAT,QVOF,PPER,EPER)
```

This routine uses three `CORPUS` routines, `TBILIN`, `TBTRPR` and `TBERPR`.
Persons Responsible: T .Adye, P. Billoir, I. Siccama

8.12 PNPVFIT

Function: Calculate the Primary Vertex for a given list of PA banks.
First all tracks with a large impact parameter are removed. Then a first iteration is done, followed by a more strict cut on the new impact parameters. The surviving tracks enter a second stage, that consists of an iterative rejection procedure. At each iteration the track with the highest χ^2 contribution, above a certain minimum, is removed.
Source Code: Deck `PNPVFIT`, Patch `REC2ND`, Cards `REC2ND`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                                  ==== ====== ====== ===== ===== ===== ======
      SUBROUTINE PNPVFIT(NDIM,NDSTPA,LDSTPA,FITWB,XBEAM,VXCUT,CHICUT,
     +    VERTEX,VERROR,CHI2VX,NDOF,CHI2TR,NT,NITER,JERR)
          ====== ====== ====== ==== ====== == ===== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`NDIM`	Number of dimensions 2 or 3, recommended value is 3
`INTEGER`	`NDSTPA`	Number of tracks to be fitted
`INTEGER`	`LDSTPA(*)`	List of PA links for tracks to be fitted
`LOGICAL`	`FITWB`	Fit with beam spot constraint or not. Recommended value is `.TRUE.`
`REAL`	`XBEAM(3)`	Coordinates of the beamspot. Warning: `XBEAM` is used only to calculate the impact parameter to select tracks for the first iteration. Inside the fit itself the contents of `/PXCPRO/` are used.
`REAL`	`VXCUT(4)`	Cuts on the impact parameter in R and z. The first two values are used in the first stage, the last two values in the second stage. Currently recommended values are: (0.25,1000.) and (0.05,1000.) respectively. For these values the the impact parameter in z is effectively not used. It is foreseen to be used for data from the new Vertex Detector, taken in 1994 and onwards.
`REAL`	`CHICUT`	Cut on the individual χ^2 contribution of a track to the fit. If there are tracks which have a contribution greater than `CHICUT`, the worst one is omitted in the next iteration. Recommended value is 5.0

Output arguments

`REAL`	`VERTEX(3)`	Coordinates of the Primary Vertex
`REAL`	`VERROR(6)`	Reduced error matrix of the vertex
`REAL`	`CHI2VX`	χ^2 of the fit
`INTEGER`	`NDOF`	Number of degrees of freedom of fit
`REAL`	`CHI2TR(*)`	Final contribution to the χ^2 for each track.
		This is negative if the track was not used in the fit :
		-1. if the blocklet `"TRAC"` is not defined (neutral)
		-2. if the track was rejected at the first stage
		-10. if the track was rejected at the second stage
`INTEGER`	`NT`	Number of tracks used in the vertex fit
`INTEGER`	`NITER`	Number of iterations used
`INTEGER`	`JERR`	= 0 no errors occurred
		= 1 no tracks to start the first iteration
		= 2 vertex fitting error in the first stage
		= 3 no tracks to start the second stage
		= 4 no tracks left in the second stage
		= 5 vertex fitting error in the second stage

Usage Notes: This is the omni–package version of the main vertex fitting routine `TBFITV`, which is used in `CORPUS`. The heart of the routine is the general vertex fitting routine `PXFVTX` (section 8.9).
Persons Responsible: P. Billoir, I. Siccama

8.13 CALNIP, CATNIP

Function: Calculate the New Impact Parameters of a track. `CALNIP` extracts the track data from the DST and calls `CATNIP`.
The standard DELPHI impact parameters are calculated relative to the DELPHI origin of coordinates. This routine calculates the impact parameters of a track relative to a new vertex, and propagates the errors. It is meant to be called after the primary vertex fitting routine `PNPVFIT` to calculate the impact parameter of a track with respect to the newly calculated primary vertex, see section (8.12).
Source Code: Deck `CALNIP`, Patch `REC2ND`, Cards `REC2ND`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                      ==== ====== ====== ====== ======
      SUBROUTINE CALNIP(NDIM,LPATRK,CHITRK,VERTEX,VERROR,
     +                  PARIMP,COVIMP,JERR)
                        ====== ====== ====

                        ==== ====   ====== ====== ======
      SUBROUTINE CATNIP(PARS,ERRS,  CHITRK,VERTEX,VERROR,
     +                  PARIMP,COVIMP,JERR)
                        ====== ====== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`NDIM`	Number of dimensions used in the vertex fit (2 or 3)
`INTEGER`	`LPATRK`	PA link of the track
`REAL`	`PARS(5)`	Perigee parameters of the track
`REAL`	`ERRS(15)`	Errors on PARS
`REAL`	`CHITRK`	Must be negative if the track was not used in the primary vertex fit (the absolute value is not used)
`REAL`	`VERTEX(3)`	Coordinates of the Primary Vertex
`REAL`	`VERROR(6)`	Reduced error matrix of the vertex

Output arguments

`REAL`	`PARIMP(3)`	Impact parameters w.r.t. the Primary Vertex. The ”perigee” being defined as the point of closest approach to a z-axis going to the fitted vertex `PARIMP(1)` is the signed distance (d.c.a. on xy projection) `PARIMP(2)` is the z of the perigee `PARIMP(3)` is the total distance of the perigee, computed from `PARIMP(1:2)` N.B. `PARIMP` is also defined if the track does not belong to the vertex.
`REAL`	`COVIMP(1:4)`	Error matrix on `PARIMP(1)`,`PARIMP(2)` accounting for errors on the fitted vertex AND the track itself, in a way depending on whether the track is included in the vertex or not. Error on `PARIMP(1)` = `SQRT(COVIMP(1))` Error on `PARIMP(2)` = `SQRT(COVIMP(3))`

		Error on `PARIMP(3)` = `SQRT(COVIMP(4))`
`INTEGER`	`JERR`	= 1 no perigee parameters found = 2 error in computing `COVIMP`

Usage Notes: Because of the track rejection procedure, the width of the distribution of `PARIMP`/error is less than 1. The elements of `COVIMP` are set to zero, if an error ocurrs.
Persons Responsible: P. Billoir, I. Siccama

9 Track Refitting

There are a few routines existing to refit tracks from their TER's at the DST level.

9.1 FK1TRK

Function: Fits a track from supplied TER's in `DOUBLE PRECISION`.
Source Code: Deck `FK1TRK`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                 ==== ====== =====       ======
SUBROUTINE FK1TRK(NTER,NDATTE,DATTE,DATTK1,DATEXT,IER1)
                                    ====== ====== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`NTER`	Number of TER's to be used (≤ 13)
`INTEGER`	`NDATTE(15)`	Number of words per TER
`REAL`	`DATTE(50,15)`	TER data
`REAL`	`DATEXT(20)`	Starting trajectory perigee parameters

Output arguments

`REAL`	`DATTK1(37)`	Fitted track in TKR format
`REAL`	`DATEXT(20)`	Fitted track perigee parameters
`INTEGER`	`IER1`	error flag; 0 = no error

Usage Notes: Debug printout is controlled by variable `IDEB` in common `/HISDEB/`. `IDEB=0` turns off printout, `IDEB=1,2` turns on successively more printout.
`TRKFIT` (section 9.2) gives a user-friendly interface to track fitting with this routine. It also will set the debug printout flag from the general DSTFIX printout flag `IFXDBG`.
Person Responsible: P. Billoir

9.2 TRKFIT, TRKFER

Function: Fits a track from supplied TER's in `DOUBLE PRECISION`.
Source Code: Deck `TRKFIT`, Patch `DSTFIX`, Cards `DSTFIX`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                  ====== ======
SUBROUTINE TRKFIT(LDSTPA,IFTASK,IFTACT,PABLK0,PABLK8,IERROR)
                                ====== ====== ====== ======


             ======
ENTRY TRKFER(LPRINT)
```

Calling Arguments:

Input arguments

`INTEGER`	`LDSTPA`	Link to track PA bank
`INTEGER`	`IFTASK`	Bitted word specifying detectors to be used
`INTEGER`	`LPRINT`	LUN for printout

Output arguments

`INTEGER`	`IFTACT`	Bitted word specifying detectors actually used
`REAL`	`PABLK0(26)`	Fitted track in DST PA bank blocklet 0 format
`REAL`	`PABLK8(20)`	Fitted track in DST PA bank blocklet 8 format (perigee parameters)
`INTEGER`	`IERROR`	error flag; 0 = no error

Usage Notes: This routine calls `FK1TRK` (section 9.1). It also ensures the material description for the detector is updated by calling `MATCOM` (section 10.1).
The bits for `IFTASK` and `IFTACT` are; bit 1 = VD, bit 2 = ID $R\Phi$ trigger, bit 4 = ID z trigger, bit 8 = ID jet chamber, bit 16 = TPC, bit 32 = OD, bit 64 = FCA and bit 128 = FCB.
Due to the change in the VD DST data format for 92 data, the VD hits used in the fit are now taken from the `VDCLAP` commons in `VDIO` (section 11). Therefore, to use VD hits in the track fit, `VDRDST` (section 11.3) must have been called in the event previous to the call to `TRKFIT`.
Debug printout is controlled (as with all DSTFIX routines) using the flag `IFXDBG` in common `/FXCOMM/`. The higher the (non-zero) value of `IFXDBG`, the more printout is generated. This also sets the internal debug flag `IDEB` for `FK1TRK` (section 9.1).
`TRKFER` can be called after all events to print a table of statistics on error values in `TRKFIT` for this job.
Person Responsible: P. Dauncey

9.3 DPAFIT

Function: Fits a track using all TER's on DST and overwrites the original DST information.
Source Code: Deck `DPAFIT`, Patch `DSTFIX`, Cards `DSTFIX`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                    ======
SUBROUTINE DPAFIT(LDSTPA)
```

Calling Arguments:

Input argument
`INTEGER` `LDSTPA` Link to track PA bank

Usage Notes: This routine calls `TRKFIT` (section 9.2) and in addition `VDRDST` (section 11.3) in order that VD hits can be used in the refit.
If the track fit failed for any reason, the original fit is left and the track is flagged as being unrefitted by setting status bit 18 in the PA bank.
WARNING: The DST information is overwritten. Also, if there was no blocklet 8 for the perigee parameters previously, then one will be created, which in general will mess up any unprotected Zebra links.
Person Responsible: P. Dauncey

9.4 DSTFIT

Function: Fits all tracks in an event and overwrites the original DST information.
Source Code: Deck `DSTFIT`, Patch `DSTFIX`, Cards `DSTFIX`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
SUBROUTINE DSTFIT
```

Calling Arguments: None
Usage Notes: This routine simply calls `DPAFIT` (section 9.3) for all tracks in the event. See the comments for that routine.
In addition, all the PV banks are compressed into one dummy PV bank before any of the tracks are refit.
Person Responsible: P. Dauncey

9.5 DSTPGE

Function: Attempts to recover perigee parameters for tracks which have no PA extra module 8 information.
Source Code: Deck `DSTPGE`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE DSTPGE
```

Calling Arguments: None
Usage Notes: This routine calls `DPAFIT` (section 9.3) for all tracks which are missing perigee parameters.
Person Responsible: P. Dauncey

10 Track Extrapolations

Described below are the EXX routines themselves, as well as some of the FKRxxx routines from CORPUS. In addition, there are FKXxxx routines which are double precision versions of the EXX and FKRxxx routines.
There are several different parameter conventions used in Delphi software. These are refered to here as the "EXX" convention, the "TKR" convention, the "TER" convention and the "perigee" convention. These are

- EXX - $(R, \Phi, z, \theta, \beta, \kappa)$ or $(z, x, y, \theta, \phi, \kappa)$
- TKR - $(R, R\Phi, z, \theta, \phi, -Q/ \mid p \mid)$ or $(x, y, z, \theta, \phi, -Q/ \mid p \mid)$
- TER - $(R, R\Phi, z, \theta, \phi, -Q/ \mid p_{xy} \mid)$ or $(x, y, z, \theta, \phi, -Q/ \mid p_{xy} \mid)$
- Perigee - $(\epsilon, z, \theta, \phi, \kappa)$

where $\beta = \phi - \Phi$ and the inverse radius of curvature is $\kappa = -QB/ \mid p_{xy} \mid = -QB/ \mid p \mid \sin\theta$. The impact parameter ϵ is normally defined relative to the origin and can be converted into an x, y position using $x = \epsilon \sin\phi$ and $y = -\epsilon \cos\phi$.

10.1 MATCOM, EXXCOM, FKXCOM, MATPRT

Function: Fills and prints commons /EXXMAT/ and /FKDDES/ with the correct material description for the event. These commons are used by EXX and FK1TRK (section 9.1).
Source Code: Deck MATCOM, Patch EXXEXTRA, Cards DSTANA, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
SUBROUTINE MATCOM


                  ======
SUBROUTINE MATPRT(ILUNIT)
```

Calling Arguments: None

Input arguments
INTEGER ILUNIT LUN for printout

Usage Notes: This routine is called from several routines which use the EXX or FK1TRK routines, for example, DSTMSP (section 10.12) and TRKFIT (section 9.2). The routine is protected against continually refilling the commons by checking for a change to the data type before the update is done. EXXCOM and FKXCOM are the actual subroutines called by MATCOM which fill the commons /EXXMAT/ and /FKDDES/ respectively. These commons use the same variable names and so cannot be filled directly in the same routine.
Note, EXX routines use the magnetic field, CONSB in GeV/cm, in common /EXXFLD/. This is also filled (from common /PXCONS/) by EXXCOM. In addition, common /FKFILD/, which contains CONSB (same variable name but in DOUBLE PRECISION) is filled by FKXCOM. This is also performed by BPILOT (section 15.3).
MATPRT prints a table of the material in common FKDDES.
Person Responsible: P. Dauncey

10.2 FKRFPE, FKXFPE, EXXPER, FKXPER, EXXPMR, FKXPMR

Function: Propagates a track from the perigee to a cylinder at a final radius, with or without multiple scattering.

Source Code: Deck `FKRFPE`, Patch `EXXX`, Cards `EXX`, Library `DSTANA`, Disk `ANA`
Deck `FKXFPE`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`
Deck `EXXPER`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`
Deck `FKXPER`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`
Deck `EXXPMR`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY` and
Deck `FKXPMR`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.

Calling Sequence:

```
                  === === === =====
SUBROUTINE FKRFPE(PAR,COV,RAD,CONSB,TSTART,IERR)
                                    ====== ====
                  === ====
SUBROUTINE FKXFPE(PAR,RADX,PARREF,IERR)
                           ====== ====
                  === === === =====
SUBROUTINE EXXPER(PAR,COV,RAD,CONSB,TPAR,TERR,IERR)
                                    ==== ==== ====
                  ==== ==== ==== ======
SUBROUTINE FKXPER(DPAR,DCOV,RADX,DCONSB,DTPAR,DTERR,IERR)
                                        ===== ===== ====
                  === === === ===== ===
SUBROUTINE EXXPMR(PAR,COV,RAD,CONSB,IMS,TPAR,TERR,IERR)
                                        ==== ==== ====
                  ==== ==== ==== ====== ===
SUBROUTINE FKXPMR(DPAR,DCOV,RADX,DCONSB,IMS,DTPAR,DTERR,IERR)
                                            ===== ===== ====
```

Calling Arguments:

Input arguments

`REAL`	`PAR(5)`	initial perigee parameters, $(\epsilon, z, \theta, \phi, \kappa)$
`REAL`	`COV(15)`	initial perigee errors
`DOUBLE PRECISION`	`DPAR(5)`	initial perigee parameters, $(\epsilon, z, \theta, \phi, \kappa)$
`DOUBLE PRECISION`	`DCOV(15)`	initial perigee errors
`REAL`	`RAD`	radius of cylinder to extrapolate to
`DOUBLE PRECISION`	`RADX`	radius of cylinder to extrapolate to
`REAL`	`CONSB`	Value of magnetic field in GeV/cm
`DOUBLE PRECISION`	`DCONSB`	Value of magnetic field in GeV/cm
`INTEGER`	`IMS`	Multiple scattering flag; 0 ignor MS, ± 1 add/subtract MS ± 2 add/subtract MS and dE/dx (not yet implemented)

Output arguments

`REAL`	`TSTART(27)`	final parameters and errors in TKR format
`DOUBLE PRECISION`	`PARREF(6)`	final parameters in EXX format

`REAL`	`TPAR(6)`	final parameters in TKR format
`REAL`	`TERR(15)`	final errors in TKR format
`DOUBLE PRECISION`	`DTPAR(6)`	final parameters in TKR format
`DOUBLE PRECISION`	`DTERR(15)`	final errors in TKR format
`INTEGER`	`IERR`	error flag; 0 = OK, 3 = no solution

Usage Notes: `FKXFPE` mixes `REAL` (`PAR`) and `DOUBLE PRECISION` (`RADX`) inputs and does not calculate errors. It takes the value of the magnetic field from `/FKFILD/`.
Persons Responsible: P. Billoir, P. Dauncey

10.3 FKRTPE, FKXTPE, EXXREP, FKXREP, EXXRMP, FKXRMP

Function: Propagates a track from a cylinder to the perigee, with or without multiple scattering.
Source Code: Deck `FKRTPE`, Patch `EXXX`, Cards `EXX`, Library `DELANAXX`, Disk `ANA`
Deck `FKXTPE`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`
Deck `EXXREP`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`
Deck `FKXREP`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`
Deck `EXXRMP`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY` and
Deck `FKXRMP`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                  ====== =====
SUBROUTINE FKRTPE(TKDATA,CONSB,PAR,COV)
                               === ===
                  ==== ====
SUBROUTINE FKXTPE(PARX,COVX,PAR,WGT,IERR)
                            === === ====
                  ==== ==== =====
SUBROUTINE EXXREP(RPAR,RCOV,CONSB,PAR,COV)
                                  === ===
                  ==== ==== ======
SUBROUTINE FKXREP(PARX,COVX,DCONSB,DPAR,DCOV)
                                   ==== ====
                  ==== ==== ===== ===
SUBROUTINE EXXRMP(RPAR,RCOV,CONSB,IMS,PAR,COV)
                                      === ===
                  ==== ==== ====== ===
SUBROUTINE FKXRMP(PARX,COVX,DCONSB,IMS,DPAR,DCOV)
                                       ==== ====
```

Calling Arguments:

Input arguments

`REAL`	`TKDATA(37)`	Initial TKR parameters
`REAL`	`CONSB`	Value of magnetic field in GeV/cm
`DOUBLE PRECISION`	`PARX(6)`	Initial TKR parameters
`DOUBLE PRECISION`	`COVX(15)`	Initial TKR errors
`REAL`	`RPAR(6)`	Initial TKR parameters
`REAL`	`RCOV(15)`	Initial TKR errors
`DOUBLE PRECISION`	`DCONSB`	Value of magnetic field in GeV/cm
`INTEGER`	`IMS`	Multiple scattering flag; 0 ignor MS, ± 1 add/subtract MS ± 2 add/subtract MS and dE/dx (not yet implemented)

Output arguments

`REAL`	`PAR(5)`	Final perigee parameters, $(\epsilon, z, \theta, \phi, \kappa)$
`REAL`	`COV(15)`	Final perigee errors
`REAL`	`WGT(15)`	Final perigee inverse errors
`INTEGER`	`IERR`	error flag; 0 = OK

DOUBLE PRECISION	DPAR(5)	Final perigee parameters, $(\epsilon, z, \theta, \phi, \kappa)$
DOUBLE PRECISION	DCOV(15)	Final perigee errors

Usage Notes: The output of FKXTPE is REAL, not DOUBLE PRECISION. It takes the value of the magnetic field from /FKFILD/.

Person Responsible: P. Billoir

10.4 EXXCYL, FKXCYL

Function: Propagates a track from a cylinder at an initial radius to a cylinder at a final radius.
Source Code: Deck EXXCYL, Patch EXXX, Cards EXX, Library DELANAXX, Disk ANA and Deck FKXCYL, Patch TKFIT, Cards TKREFIT, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                  ==== ==== ==== === === ====== ====
SUBROUTINE EXXCYL(PARI,IDIR,RADF,ZMN,ZMX,SINBMX,IOPT,PARF,DER,ALRPHI,IERR)
                                                     ==== === ====== ====
```

Calling Arguments:

Input arguments

REAL	PARI(6)	initial parameters $(R, \Phi, z, \theta, \beta, \kappa)$
INTEGER	IDIR	direction flag; 1 = positive, -1 = negative, 0 = both
REAL	RADF	radius of cylinder to extrapolate to
REAL	ZMN	lower z limit of the cylinder
REAL	ZMX	upper z limit of the cylinder
REAL	SINBMX	maximum allowed $\|\sin\beta\|$
INTEGER	IOPT	option flag; 0 = no derivatives, 1 = derivatives calculated

Output arguments

REAL	PARF(6)	final parameters (as for PARI)
REAL	DER(8)	derivatives of final w.r.t. initial parameters
REAL	ALRPHI	length in $R\Phi$ plane of extrapolation
INTEGER	IERR	error flag; 0 = OK, 1 = no intersection, 2 = SINBMX exceeded, 3 = outside z limits

Usage Notes: FKXCYL has the same calling sequence as EXXCYL, but all REAL arguments are now DOUBLE PRECISION.
The parameters must be supplied in "EXX" convention, that is $(R, \Phi, z, \theta, \beta, \kappa)$.
Persons Responsible: J. Wickens, P. Billoir

10.5 EXXDCY, FKXDCY

Function: Calculates derivatives for propagation of a track from a cylinder at an initial radius to a cylinder at a final radius, when both sets of parameters are known.
Source Code: Deck `EXXDCY`, Patch `EXXX`, Cards `EXX`, Library `DELANAXX`, Disk `ANA` and Deck `FKXDCY`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                    ====== ======
SUBROUTINE EXXDCY(PARAMI,PARAMF,DER)
                                ===
```

Calling Arguments:

Input arguments

`REAL`	`PARAMI(6)`	initial parameters $(R, \Phi, z, \theta, \beta, \kappa)$
`REAL`	`PARAMF(6)`	final parameters

Output argument

`REAL`	`DER(8)`	derivatives of final w.r.t. initial parameters

Usage Notes: FKXDCY has the same calling sequence as EXXDCY, but all `REAL` arguments are now `DOUBLE PRECISION`.
The parameters must be supplied in “EXX” convention, that is $(R, \Phi, z, \theta, \beta, \kappa)$.
Persons Responsible: J. Wickens, P. Billoir

10.6 EXXCVC

Function: Propagates a track covariance matrix from a cylinder at an initial radius to a cylinder at a final radius.
Source Code: Deck `EXXCVC`, Patch `EXXX`, Cards `EXX`, Library `DELANAXX`, Disk `ANA` and Deck `FKXCVC`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                  === ===
SUBROUTINE EXXCVC(CVI,DER,CVF)
                          ===
```

Calling Arguments:

Input arguments

`REAL`	`CVI(15)`	Initial covariance matrix
`REAL`	`DER(8)`	Derivatives of final w.r.t. initial parameters

Output argument

`REAL`	`CVF(15)`	Final covariance matrix

Usage Notes: FKXCVC has the same calling sequence as EXXCVC, but all `REAL` arguments are now `DOUBLE PRECISION`.
Persons Responsible: J. Wickens, P. Billoir

10.7 EXXPLA, FKXPLA

Function: Propagates a track from a plane at an initial z to a plane at a final z.
Source Code: Deck `EXXPLA`, Patch `EXXX`, Cards `EXX`, Library `DELANAXX`, Disk `ANA` and Deck `FKXPLA`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                  ====== ==== == === === ====
SUBROUTINE EXXPLA(PARAMI,IDIR,ZF,RMN,RMX,IOPT,PARAMF,DER,ALRPHI,IERR)
                                              ====== === ====== ====
```

Calling Arguments:

Input arguments

`REAL`	`PARAMI(6)`	initial parameters $(z, x, y, \theta, \phi, \kappa)$
`INTEGER`	`IDIR`	direction flag; 1 = positive, -1 = negative, 0 = both
`REAL`	`ZF`	z of plane to extrapolate to
`REAL`	`RMN`	lower radius limit of the plane
`REAL`	`RMX`	upper radius limit of the plane
`INTEGER`	`IOPT`	option flag; 0 = no derivatives, 1 = derivatives calculated

Output arguments

`REAL`	`PARAMF(6)`	final parameters (as for `PARAMI`)
`REAL`	`DER(8)`	derivatives of final w.r.t. initial parameters
`REAL`	`ALRPHI`	length in $R\Phi$ plane of extrapolation
`INTEGER`	`IERR`	error flag; 0 = OK, 1 = no intersection, 2 = `SINBMX` exceeded, 3 = outside z limits

Usage Notes: FKXPLA has the same calling sequence as EXXPLA, but all `REAL` arguments are now `DOUBLE PRECISION`.
The parameters must be supplied in “EXX” convention, that is $(z, x, y, \theta, \phi, \kappa)$.
Persons Responsible: J. Wickens, P. Billoir

10.8 EXXDPL, FKXDPL

Function: Calculates derivatives for propagation of a track from a plane at an initial z to a plane at a final z, when both sets of parameters are known.
Source Code: Deck `EXXDPL`, Patch `EXXX`, Cards `EXX`, Library `DELANAXX`, Disk `ANA` and Deck `FKXDPL`, Patch `TKFIT`, Cards `TKREFIT`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                  ====== ======
SUBROUTINE EXXDPL(PARAMI,PARAMF,DER)
                                ===
```

Calling Arguments:

Input arguments

`REAL`	`PARAMI(6)`	initial parameters $(z, x, y, \theta, \phi, \kappa)$
`REAL`	`PARAMF(6)`	final parameters

Output argument

`REAL`	`DER(8)`	derivatives of final w.r.t. initial parameters

Usage Notes: FKXDPL has the same calling sequence as EXXDPL, but all `REAL` arguments are now `DOUBLE PRECISION`.
The parameters must be supplied in "EXX" convention, that is $(z, y, x, \theta, \phi, \kappa)$.
Persons Responsible: J. Wickens, P. Billoir

10.9 EXXCVP

Function: Propagates a track covariance matrix from a plane at an initial z to a plane at a final z,
Source Code: Deck EXXCVP, Patch EXXX, Cards EXX, Library DELANAXX, Disk ANA and Deck FKXCVP, Patch TKFIT, Cards TKREFIT, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                  === ===
SUBROUTINE EXXCVP(CVI,DER,CVF)
                          ===
```

Calling Arguments:

Input arguments

REAL	CVI(15)	Initial covariance matrix
REAL	DER(8)	Derivatives of final w.r.t. initial parameters

Output argument

REAL	CVF(15)	Final covariance matrix

Usage Notes: FKXCVP has the same calling sequence as EXXCVP, but all REAL arguments are now DOUBLE PRECISION.
Persons Responsible: J. Wickens, P. Billoir

10.10 EXXCBF, EXXRTZ, FKXRTZ, EXXPTZ,FKXPTZ

Function: Converts parameters and errors at a point from cylindrical or perigee to plane definitions, without propagation.
Source Code: Deck `EXXCBF`, Patch `EXXX`, Cards `EXX`, Library `DELANAXX`, Disk `ANA` and Deck `EXXRTZ`, Patch `EXXEXTRA`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                  ==== ===
SUBROUTINE EXXCBF(PARB,CVB,PARF,CVF)
                           ==== ===
                  ==== ==== =====
SUBROUTINE EXXRTZ(PARR,COVR,CONSB,PARZ,COVZ)
                                  ==== ====
                  ===== ===== ======
SUBROUTINE FKXRTZ(DPARR,DCOVR,DCONSB,DPARZ,DCOVZ)
                                     ===== =====
                  ==== ==== =====
SUBROUTINE EXXPTZ(PERR,PCVR,CONSB,PARZ,COVZ)
                                  ==== ====
                  ===== ===== ======
SUBROUTINE FKXPTZ(DPERR,DPCVR,DCONSB,DPARZ,DCOVZ)
                                     ===== =====
```

Calling Arguments:

Input arguments

`REAL`	`PARB(6)`	Cylindrical parameters $(R, \Phi, z, \theta, \beta, \kappa)$
`REAL`	`CVB(15)`	Cylindrical covariance matrix
`REAL`	`PARR(6)`	Cylindrical TKR parameters $(R, R\Phi, z, \theta, \phi, -Q/p)$
`REAL`	`COVR(15)`	Cylindrical TKR covariance matrix
`REAL`	`CONSB`	Value of magnetic field in GeV/cm
`DOUBLE PRECISION`	`DPARR(6)`	Cylindrical TKR parameters $(R, R\Phi, z, \theta, \phi, -Q/p)$
`DOUBLE PRECISION`	`DCOVR(15)`	Cylindrical TKR covariance matrix
`DOUBLE PRECISION`	`DCONSB`	Value of magnetic field in GeV/cm
`REAL`	`PERR(6)`	Perigee parameters $(\epsilon, z, \theta, \phi, \kappa)$
`REAL`	`PCVR(15)`	Perigee covariance matrix
`DOUBLE PRECISION`	`DPERR(6)`	Perigee parameters $(\epsilon, z, \theta, \phi, \kappa)$
`DOUBLE PRECISION`	`DPCVR(15)`	Perigee covariance matrix

Output arguments

`REAL`	`PARF(6)`	Planar parameters $(z, x, y, \theta, \phi, \kappa)$
`REAL`	`CVF(15)`	Planar covariance matrix
`REAL`	`PARZ(6)`	Planar TKR parameters $(x, y, z, \theta, \phi, -Q/p)$
`REAL`	`COVZ(15)`	Planar covariance matrix
`DOUBLE PRECISION`	`DPARZ(6)`	Planar TKR parameters $(x, y, z, \theta, \phi, -Q/p)$
`DOUBLE PRECISION`	`DCOVZ(15)`	Planar covariance matrix

Usage Notes:
Persons Responsible: J. Wickens, P. Billoir, P. Dauncey

10.11 EXXCFB

Function: Converts parameters and errors at a point from plane to cylindrical definitions, without propagation.
Source Code: Deck EXXCFB, Patch EXXX, Cards EXX, Library DELANAXX, Disk ANA and Deck FKXCFB, Patch TKFIT, Cards TKREFIT, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                  ==== ===
SUBROUTINE EXXCFB(PARF,CVF,PARB,CVB)
                           ==== ===
```

Calling Arguments:

Input arguments

REAL	PARF(6)	Planar parameters $(z, x, y, \theta, \phi, \kappa)$
REAL	CVF(15)	Planar covariance matrix

Output arguments

REAL	PARB(6)	Cylindrical parameters $(R, \Phi, z, \theta, \beta, \kappa)$
REAL	CVB(15)	Cylindrical covariance matrix

Usage Notes: FKXCFB has the same calling sequence as EXXCFB, but all REAL arguments are now DOUBLE PRECISION.
Persons Responsible: J. Wickens, P. Billoir

10.12 DSTMSP

Function: Corrects perigee parameter errors for multiple scattering not included in DST, for all charged tracks in the event.
Source Code: Deck `DSTMSP`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
SUBROUTINE DSTMSP
```

Calling Arguments: None
Usage Notes: This routine simple loops over all PA banks and calls `DPAMSP` (section 10.13). It also ensures the material description for the detector is updated by calling `MATCOM` (section 10.1).
Person Responsible: P. Dauncey

10.13 DPAMSP

Function: Corrects perigee parameter errors for multiple scattering not included in DST, for a single charged track.
Source Code: Deck `DPAMSP`, Patch `DSTIO`, Cards `DSTANA`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                  ======
SUBROUTINE DPAMSP(LDSTPA)
```

Calling Arguments:

Input argument

`INTEGER`	`LDSTPA`	Link to PA bank

Usage Notes: The bit 14 of the PA bank status word is used to flag this correction having been performed. Therefore, on a second or further call for the same track, `DPAMSP` will NOT add on a further amount to the error matrix, but will simply return.
To correct all the tracks in an event, use `DSTMSP` (section 10.12).
Person Responsible: P. Dauncey

10.14 FKMI5, FKMI4, FKMI3, FKDI5, FKDI4, FKDI3

Function: Inverts a 5×5, 4×4 or 3×3 weight or error matrix in triangular form.
Source Code: Decks FKMIn, Patch PXRFIT, Cards CORPUSXX, Library CORPUSXX, Disk TEAM11 and Decks FKDIn, Patch TKFIT, Cards TKREFIT, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                    ===
SUBROUTINE FKMIn(WGT,COV,IERR)
                         === ====
```

Calling Arguments:

Input argument

REAL	WGT(n(n+1)/2)	Input weight or error matrix

Output arguments

REAL	WGT(n(n+1)/2)	Inverted matrix
INTEGER	IERR	error flag; 0 = no error, 2 = not positive

Usage Notes: FKMIn invertes an $n \times n$ matrix. FKDIn has an identical calling sequence but has DOUBLE PRECISION matrices for input and output.
COV may overwrite WGT.
Person Responsible: P. Billoir

11 Vertex Detector Routines

VDCLAP is a set of routines for manipulating the VD information via a large common /VDIO/ (in VDKEEP CARDS on the MVTX disk, which should be consulted for details). It is mainly intended for use by VD experts, but non-experts can use the routines as "black boxes" to do various tasks without needing to know the detailed structure of /VDIO/. VDCLAP contains many more routines than the few listed here, but these cover most of what is usable by the non-expert.

11.1 READCF

Function: Reads in a VD alignment file. Up to 5 can be read in per job.
Source Code: Deck READCF, Patch VDALIGN, Cards VDALIGN, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                  ===== ======
SUBROUTINE READCF(LUNIN,LUNOUT)
```

Calling Arguments:

Input arguments

INTEGER	LUNIN	Logical unit number for alignment file
INTEGER	LUNOUT	Logical unit number for comments printout

Usage Notes: The required alignment file is always refered to by the unit number used to read it in. For example, EVASSO (section 11.5) must be given the unit number of the alignment file to use as an input argument.
VDMOVE (section 11.2) can be used to give an overall shift to the alignment.
Person Responsible: H. Dijkstra

11.2 VDMOVE

Function: Shifts a VD alignment file by an arbitrary global transformation for each half-shell (B and D) separately.
Source Code: Deck VDMOVE, Patch VDALIGN, Cards VDALIGN, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                  ==== ==== ===== ===== ==== ==== ==== ====
SUBROUTINE VDMOVE(LUN1,LUN2,DIRCB,DIRCD,ANGB,ANGD,XYZB,XYZD,IERR)
                                                            ====
```

Calling Arguments:

Input arguments

INTEGER	LUN1	Logical unit number for original alignment file
INTEGER	LUN2	Logical unit number for new alignment
REAL	DIRCB(3)	Vector for half-shell B to rotate around
REAL	DIRCD(3)	Vector for half-shell D to rotate around
REAL	ANGB	Rotation angle for half-shell B
REAL	ANGD	Rotation angle for half-shell D
REAL	XYZB(3)	Shift in x,y,z for half-shell B
REAL	XYZD(3)	Shift in x,y,z for half-shell D

Output argument

INTEGER	IERR	Error flag; 0 = OK

Usage Notes: The required alignment file must be first read in by READCF (section 11.1).
Person Responsible: W. Murray

11.3 VDIDST, VDRDST, VDWDST, VDEDST

Function: Unpacks VD and track information into VDCLAP commons so that other VDCLAP routines can be used.
Source Code: Decks `VDIDST, VDRDST, VDEDST`, Patch `VDIODST`, Cards `VDIODST`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
SUBROUTINE VDIDST

SUBROUTINE VDRDST

SUBROUTINE VDWDST

SUBROUTINE VDEDST
```

Calling Arguments: None
Usage Notes: `VDIDST` should be called once at initialisation; note this is done by the standard Delphi initialisation routine `DSTBEG` (section 3.1). `VDRDST` is the actual routine which does the VD data unpacking and should be called once per event. `VDWDST` updates the DST information with any changes which have been made via the `VDCLAP` common and should therefore be used with great care (it is also not yet fully complete). `VDEDST` reports on errors found in the job and should be called after all events have been processed.
To use the VD information in the track refit, `TRKFIT` (section 9.2), `VDRDST` must have been called for the event.
Person Responsible: T. Adye

11.4 VDISIM, VDRSIM, VDESIM

Function: Unpacks VD information and simulation truth tracks into VDCLAP commons so that other VDCLAP routines can be used.
Source Code: Decks `VDISIM, VDRSIM, VDESIM`, Patch `VDIODST`, Cards `VDIODST`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
SUBROUTINE VDISIM
SUBROUTINE VDRSIM
SUBROUTINE VDESIM
```

Calling Arguments: None
Usage Notes: `VDISIM` should be called once at initialisation. `VDRSIM` is the actual routine which does the VD data unpacking and should be called once per event. `VDESIM` reports on errors found in the job and should be called after all events have been processed.
Person Responsible: T. Adye

11.5 EVASSO, EVAOTD, EVAOTK

Function: Associates VD points with Delphi tracks and stores the result in the VDCLAP common blocks.
Source Code: Decks `EVASSO, EVAOTD, EVAOTK`, Patch `EVASSO`, Cards `EVASSO`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                   ======
SUBROUTINE EVASSO(LUNALI)
SUBROUTINE EVAOTD
SUBROUTINE EVAOTK
```

Calling Arguments:

Input argument
`INTEGER` `LUNALI` Unit number of VD alignment file

Usage Notes: This is the 911101 version, which works for all single-sided geometries (two and three layers). The next version will also do double-sided geometries. EVASSO takes its input from the VDCLAP common blocks, which are filled by VDRDST (section 11.3). A track can have up to two associated VD points per layer (in overlaps). A point is never shared by two or more tracks.
LUNALI is the unit number of the VD alignment file as given to READCF (section 11.1). It is an identifier. Only READCF actually reads the file.
EVAOTD and EVAOTK should be called immediately after EVASSO to get the result stored in the VDCLAP common blocks. This will be done automatically in the next version.
EVASSO is called by DELANA, and the associated VD points are normally included in the final track fit. Nevertheless, it might be useful to call EVASSO again. It should be noted that single-layer associations are unreliable. Most analyses require tracks to have hits in at least two VD layers.
Person Responsible: M. Karlsson

11.6 EVATAN

Function: Gives the result of EVASSO per track. Used by people who do not look at the VDCLAP common blocks. The track and the VD points are identified by their Tanagra labels.
Source Code: Deck `EVATAN`, Patch `EVASSO`, Cards `EVASSO`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                          ===
SUBROUTINE EVATAN( ITK, ITD, N, IQ)
                                 ===  =  ==
```

Calling Arguments:

Input argument

`INTEGER`	`ITK`	Tanagra label of a track

Output arguments

`INTEGER`	`ITD(6)`	Labels of the associated VD points
`INTEGER`	`N`	Number of associated points
`INTEGER`	`IQ`	Association "quality index"

Usage Notes: With the new VD DST format, the Tanagra labels of VD points are no longer available. In this case, ITD contains the sequential indices of the VD points (as they appear on the DST). This is taken care of by VDRDST (section 11.3) and is transparent to EVATAN.
The "quality index" is not very useful.
Person Responsible: M. Karlsson

11.7 GREVT

Function: Draws a picture of the information in the Vertex Detector common block, `VDIO`.
Source Code: Deck `GREVT`, Patch `GRMAIN`, Cards `GRACOM`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                 ====== ===== === ======
SUBROUTINE GREVT(FNAMIN,TRACK,OPT,LUNINP)
```

Calling Arguments:

Input arguments

`CHARACTER*40`	`FNAMIN`	The filename to use for output.
`INTEGER`	`TRACK`	Which track to concentrate on. 0 = whole event.
`CHARACTER`	`OPT`	Various drawing options.
`INTEGER`	`LUNINP`	The unit number of the Microvertex alignment file.

Usage Notes: This routines makes either a PostScript or a GKS (not recommended) metafile which contains a picture of the vertex detector, it hits, and the charged tracks passing through it. If `FNAMIN` is set to '-', it makes up a file name based on the event number, which reduces the chance of clashes. `TRACK` controls how the event s drawn. 0 means the whole VD, positive is assumed to be the number of a track, and a view of its intersection with the silicon is shown. –1 requests a blow up of the area around the beam spot.
The plotting options are:

a	Mark the VD hits at the positions given in the VD common.
c	Produce a colour metafile, if using PostScript.
d	Do a double picture output, suitable for presentations.
f	Force the tracks through their associated VD hits.
g	Make a GKS metafile, not a PostScript one.
h	Write the VD hit numbers next to each hit.
t	Do not use the refit area of the VD common. Default is to use it if available.
u	Code the track colour and line styles according to user supplied list.
v	Draw secondary vertices, extrapolate tracks to these. Contact the author if this option is required.

For example, the "standard" VD event picture is therefore given by

```
CALL GREVT('-',0,' ',LUNINP)
```

Person Responsible: W. Murray

11.8 VDREFI

Function: Refits a track which has not had the VD information used to inlcude those hits linked in the VD common.
Source Code: Deck `VDREFI`, Patch `VDTKUSE`, Cards `VDTKUSE`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                    ======= =====
SUBROUTINE VDREFI(SILRES,LUNIN)
```

Calling Arguments:

Input arguments

`REAL`	`SILRES`	The resolution for the VD hits, cm
`INTEGER`	`LUNIN`	The unit number of the Microvertex alignment file.

Usage Notes: This routine is just an interface to `FKRTPE`, which actually performs the refit. It merely makes the calling sequence easier for VD Common block users.
The fitted parameters are put into the VDCLAP common.
Note that the author has not used this routine for some years.
Person Responsible: W. Murray

11.9 VDPREV

Function: Prints out event information (mostly from /VDIOEV/) to unit LUNPRT.
Source Code: Deck VDPREV, Patch VDIOMISC, Cards VDIOMISC, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                     ======
SUBROUTINE VDPREV (LUNPRT)
```

Calling Arguments:

Input argument
INTEGER LUNPRT Unit number to write to

Usage Notes: VDPREV(6) is called automatically by VDRDST (section 11.3) and the equivalent routines for n-tuples (etc), when the debug level (IVDBUG) is 2 or greater.
Person Responsible: T. Adye.

11.10 VDPRHT, VDPRTD, VDPRTR, VDPRTK

Function: Prints out VD hit information from `/VDIOTD/` and track information from `/VDIOTK/`.
Source Code: Deck `VDPRTD`, Patch `VDIOMISC`, Cards `VDIOMISC`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                      ======
SUBROUTINE VDPRHT (LUNPRT)

                      ======
SUBROUTINE VDPRTD (LUNPRT)

                      ======
SUBROUTINE VDPRTR (LUNPRT)

                      ======
SUBROUTINE VDPRTK (LUNPRT)
```

Calling Arguments:

Input argument

`INTEGER`	`LUNPRT`	Unit number to write to

Usage Notes: `VDPRHT` and `VDPRTD` print VD hit information from `VDIOTD` (section 2.8), while `VDPRTR` and `VDPRTK` print track information from `VDIOTK` (section 2.8).
`VDPRHT` and `VDPRTR` give a formatted summary (one line per hit or track), while `VDPRTD` and `VDPRTK` show everything in `VDIO` in horrendous detail with no real attempt at clarity. They are probably best used only by experts.
`VDPRHT(6)` and `VDPRTR(6)` are called automatically by `VDRDST` (section 11.3) and the equivalent routines for n-tuples (etc), when the debug level (`IVDBUG`) is 2 or greater. It also calls `VDPRTD(6)` and `VDPRTK(6)` when the debug level is 3 or greater.
Person Responsible: T. Adye.

11.11 VDC2X

Function: Converts a (`REAL`) channel number to x_{local}.
Source Code: Deck `VDC2X`, Patch `VDIOMISC`, Cards `VDIOMISC`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                        ====== ==== =====
REAL FUNCTION VDC2X(MODULE,CHAN,IVD90,IERR)
              =====                  ====
```

Calling Arguments:

Input arguments

`INTEGER`	`MODULE`	Module number = $(\text{sign}Z) \times (100(\text{layer} - 1) + \text{sector})$
`REAL`	`CHAN`	Real channel number
`INTEGER`	`IVD90`	Year number in VD convention

Outputs

`REAL`	`VDC2X`	x_{local} coordinate
`INTEGER`	`IERR`	Error flag: 0=OK, 1=`CHAN`$<$min, 2=`CHAN`$>$max

Person Responsible: T. Adye.

11.12 VDX2C

Function: Converts x_{local} to a (`REAL`) channel number.
Source Code: Deck `VDX2C`, Patch `VDIOMISC`, Cards `VDIOMISC`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
                 ====== ====== =====
FUNCTION VDX2C(MODULE,XLOCAL,IVD90,IERR)
         =====                     ====
```

Calling Arguments:

Input arguments

`INTEGER`	`MODULE`	Module number: $(\text{sign}Z) \times (100(\text{layer} - 1) + \text{sector})$
`REAL`	`XLOCAL`	x_{local} coordinate
`INTEGER`	`IVD90`	Year number in VD convention

Outputs

`REAL`	`VDX2C`	Real channel number
`INTEGER`	`IERR`	Error flag: 0=OK, 1=`XLOCAL`$<$min, 2=`XLOCAL`$>$max

Usage Notes:
`VDX2C` is implemented as an entry point within `VDC2X` (section 11.11) so it can use the same internal arrays.
`VDX2C` is used when the `VDCLAP COMMON`s are filled in order to fill `VDTD(IVHRST,*)`.
Person Responsible: T. Adye.

11.13 VDDLH1

Function: Cuts the specified hits from the event, removing them from the VDTD array, and adjusting the associated hit numbers in the VDTK array.
Source Code: Deck VDDLH1, Patch VDIOMISC, Cards VDIOMISC, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                 ===== ==== ======
SUBROUTINE VDDLH1(IHITS,MASK,MINASS)
```

Calling Arguments:

Input arguments

INTEGER	IHITS(*)	Array of hit numbers to be removed from VDTD, with the last element zero
INTEGER	MASK	Mask of bits to be compared with IVDTD(IVHBAD,*)
INTEGER	MINASS	If MINASS ≥ 1, then hits without an associated track will be removed

Usage Notes: This routine physically removes the hit data from the VDTD array (shuffling up the other hits into the space created). It may be more convenient to mark the hits as 'bad' by setting a bit in VDTD(IVHBAD,*) and ensuring that the same bit is set in MASKTD (in /VDGENS/). When this is done all the VD routines should ignore this hit.
Person Responsible: T. Adye.

11.14 VDDLT1

Function: Cuts the specified tracks from the event, removing them from the VDTK array, and adjusting the associated track numbers in the VDTD array.
Source Code: Deck VDDLT1, Patch VDIOMISC, Cards VDIOMISC, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                          ===== ==== ====== ====== ====== ==== ====
SUBROUTINE VDDLT1(ITRKS,MASK,MINASS,IMPMAX,THEMIN,PMIN,PMAX)
```

Calling Arguments:

Input arguments

INTEGER	ITRKS(*)	Array of track numbers to be removed from VDTK, with the last element zero
INTEGER	MASK	Mask of bits to be compared with IVDTK(IVTBAD,*)
INTEGER	MINASS	Specifies the minimum number of associated hits for the track to not be cut
REAL	IMPMAX	Maximum impact parameter for track to be accepted
REAL	THEMIN	Minimum track polar angle
REAL	PMIN	Minimum momentum.
REAL	PMAX	Maximum momentum.

Usage Notes: If any of the parameters is specified as zero, then no cut will be made on that parameter. This routine physically removes the track data from the VDTK array (shuffling up the other tracks into the space created). It may be more convenient to mark the tracks as 'bad' by setting a bit in VDTK(IVTBAD,*) and ensuring that the same bit is set in MASKTK (in /VDGENS/). When this is done all the VD routines should ignore these tracks.
Person Responsible: T. Adye.

11.15 VDCUTD

Function: Label bad VD hits in `VDTD(IVHBAD,*)`. Assumes the labels have been zeroed previously; i.e. does not overwrite any existing bad channel information.
Source Code: Deck `VDCUTD`, Patch `VDIOMISC`, Cards `VDIOMISC`, Library `VDCLAP`, Disk `MVTX`.
Calling Sequence:

```
SUBROUTINE VDCUTD
```

Calling Arguments: None
Usage Notes: This routine is called automatically by `VDRDST` (section 11.3) and the equivalent routines for n-tuples (etc), but can be used independently.
Person Responsible: P. Dauncey.

11.16 VDCUTK

Function: Label bad VD tracks in VDTK(IVTBAD,*). Assumes the labels have been zeroed previously; i.e. does not overwrite any existing bad channel information.
Source Code: Deck VDCUTK, Patch VDIOMISC, Cards VDIOMISC, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
SUBROUTINE VDCUTK
```

Calling Arguments: None
Usage Notes: VDCUTK is called automatically by VDRDST (section 11.3) and the equivalent routines for n-tuples (etc), but can be used independently.
This routine does nothing at the moment.
Person Responsible: P. Dauncey.

11.17 VDLASH

Function: Numbers the laser hits in the data.
Source Code: Deck VDLASH, Patch VDIOMISC, Cards VDIOMISC, Library VDCLAP, Disk MVTX.
Calling Sequence:

```
                  =====
SUBROUTINE VDLASH(JVDTD,ILASER,DELTAX)
                        ====== ======
```

Calling Arguments:

Input argument

INTEGER	JVDTD	Hit number in VDTD

Output arguments

INTEGER	ILASER	Laser identifier
REAL	DELTAX	Distance from nominal laser position

Persons Responsible: P. Dauncey, W. Trischuk

12 Fixing Routines

From DSTFIX area.

12.1 DSTFIX

Function: Does fixes for all detectors available, as requested through common.
Source Code: Deck `DSTFIX`, Patch `DSTFIX`, Cards `DSTFIXXX`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
SUBROUTINE DSTFIX
```

Calling Arguments: None
Usage Notes: `DSTFIX` reads in several files on pre-defined LUN's. These include VD alignment files and TPC correction constants. The program will normally fail if these files are not assigned correctly. Cantact the Person Responsible if you need to run `DSTFIX` in your own analysis. Also, see DSTFIXxx CARDS for more information.
Person Responsible: P. Dauncey

12.2 PIDTRA

Function: Transforms an ID TER from the ID coordinate system to the Delphi system.
Source Code: Deck `PIDTRA`, Patch `PIDTRA`, Cards `PIDTRA`, Library `DSTFIXXX`, Disk `DSTFIX`.
Calling Sequence:

```
                  ==== ==== ==
SUBROUTINE PIDTRA(NRUN,IEVT,TK,TEID,IERR)
                               ==== ====
```

Calling Arguments:

Input arguments

`INTEGER`	`NRUN`	Run number
`INTEGER`	`IEVT`	Event number
`REAL`	`TK(5)`	Approximate perigee parameters

Output arguments

`REAL`	`TEID(6)`	ID TER parameters in Delphi frame
`INTEGER`	`IERR`	Error flag: 0 if no error

Usage Notes: This routine must be called before any use is made of an ID TER in the Delphi frame. `TRKFIT` (section 9.2) automatically calls `PIDTRA` before using any ID TER in a track fit.
Persons Responsible: T. Adye, J. Timmermanns

12.3 PXVCTP, PXUCTP92

Function: Does fixes for 91D, 91E, 91F or 92D TPC TER's.
Source Code: Deck `PXVCTP`, Patch `TPCFIX`, Cards `TPCFIX`, Library `DSTANA`.
Calling Sequence:

```
                  ====== ======
SUBROUTINE PXVCTP(DATTPC,IDLANA)
                  ======
                    ====== ======
SUBROUTINE PXUCTP92(DATTPC,IDLANA)
                    ======
```

Calling Arguments:

Input arguments

`REAL`	`DATTPC(26)`	Initial TPC TER
`INTEGER`	`IDLANA`	Type of data; 1 = 91D, 2 = 91E/F

Output argument

`REAL`	`DATTPC(26)`	Final TPC TER

Usage Notes: `PXVCTP` is for 91 data, `PXUCTP92` for 92D.
The call sequence is changed compared with `PXUCTP`. Previously, `IDLANA` was `BZFLD`, which was unused.
`PXVCTP` reads in a constants file `CORDZ1 DAT` on the first call, using LUN 51. `PXUCTP92` reads in constants files `STAB92 CAR` and `VDTCP92 CAR` on the first call, using LUN 52 and 53.
Person Responsible: P. Roudeau

12.4 DSFHPC

Function: Performs fixes for the HPC information for charged tracks only !!!.
Source Code: Deck DSFHPC, Patch DSFHPC, Cards DSFHPC, Library DSTANA, Disk ELECID.
Calling Sequence:

```
                   ===== ==== ====
SUBROUTINE DSFHPC(CYEAR,LLPA,IACT,IERR)
                                  ====
```

Calling Arguments:

Input arguments

CHARACTER*4	CYEAR	Reprocessing (91E, 91F, 92C or 92D)
INTEGER	LLPA	PA identifier of track to be considered
INTEGER	IACT	Action flag (see below)

Output argument

INTEGER	IERR	Error flag; 0 no error, < 0 if no HPC shower.

Usage Notes: The action flag IACT should be set to 0 for fixes to be written over the PA bank information, 1 for the fixes to be done but stored in a common, and 2 for no fixes to be done and the common filled with the original information.
This routine is called by ELPROB (section 6.1) and so the user does not normally need to call DSFHPC explicitly.
There is no protection against multiple calls, which will therefore result in the fixes being done many times. It is therefore recommended never to use the option IACT=0, so that the original data are always left in the DST structure.
Persons Responsible: S. Schael

12.5 RMUCFT (MUCFIX)

Function: The calling routine for MUCFIX: for a specified track with an existing muon blocklet, the muon blocklet is refitted with a set of systematic cures/user criteria active.
Source Code: Deck `RMUCFT`, Patch `REFDST` in `*MUCFIX`, Cards `MUCFIX`, Library `DSTANA`, Disk `TEAM11`.
Calling Sequence:

```
                      === ====== ======
SUBROUTINE RMUCFT(IPA,RINITL,ISTEER,RREFIT,IERREF)
                                    ====== ======
```

Calling Arguments:

Input arguments

`INTEGER`	`IPA`	PA identifier of track to be considered
`REAL`	`RINITL(99)`	Array holding existing muon blocklet; elements stored sequentially up to maximum of 99
`INTEGER`	`ISTEER(30)`	Array controlling the systematic cure/ user feature options to be activated in the refit

Output arguments

`REAL`	`RREFIT(99)`	Array holding refitted muon blocklet; elements stored sequentially up to maximum of 99
`INTEGER`	`IERREF`	Error flag: 0 for successful refit; 1 for no fit made; 2 for error

Usage Notes:
For more information on MUCFIX and the correspondence between ISTEER elements and the various systematic cure/user feature options, see DELPHI 93–13 PHY 262.
Persons Responsible: G. Wilkinson

12.6 PFXBIT, GFXBIT, WFXBIT

Function: Set and interpret bits to indicate performed fixes.
Source Code: Deck PFXBIT, GFXBIT, WFXBIT, Patch DSTFIX, Cards DSTFIX, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                      =====
SUBROUTINE PFXBIT(CHFIX)

                             =====
LOGICAL FUNCTION GFXBIT(CHFIX)

                      ======
SUBROUTINE WFXBIT(ILUNIT)
```

Calling Arguments:

Input arguments

CHARACTER*4	CHFIX	Type of fix information required
INTEGER	ILUNIT	LUN for printout

Output

LOGICAL	GFXBIT	Flag for whether fixes performed or not

Usage Notes: PFXBIT sets the bit for the fixes corresponding to CHFIX. This should only be done by the experts within the fixing code. GFXBIT can be called by a user to determine if the bit is set (and hence the fix done); if GFXBIT=.TRUE., then the fixes have been performed. WFXBIT will print a small table of the status of the fix bits to unit ILUNIT.
The types of fixes currently defined are:

```
CHFIX = 'VD  ', 'ID  ', 'TPC ', 'OD  ', 'FCA ', 'FCB ', 'RIB ', 'RIF ',
        'HPCC', 'HPCN', 'EMFC', 'EMFN', 'HACB', 'HACF', 'MUB ', 'MUF '
```

Person Responsible: P. Dauncey

13 Simulation Truth Information

13.1 SIMGEN

Function: Prints out the information from the begin run record of the simulation parameters used for a simulation dataset. Also checks for significant changes in parameters between different tapes.
Source Code: Deck `SIMGEN`, Patch `SIMULATION`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                  ====== ====== ======
SUBROUTINE SIMGEN(LSTRUN,ILUNIT,IPRINT)
```

Calling Arguments:

Input arguments

`INTEGER`	`LSTRUN`	link to start of begin-run record data
`INTEGER`	`ILUNIT`	logical unit number for printout
`INTEGER`	`IPRINT`	Print level; 0 = errors only, 1 = basic information, 2 = full information

Usage Notes: This routine only works when the start-of-run record is present, so it cannot be called for a normal event. The place to call this routine therefore depends on which analysis package the user is using. However, all packages should have the link defined as `LSTRUN` in common `/UXLRUN/` (section 2.3).
Person Responsible: P. Dauncey

13.2 SHTREE, SHLIST

Function: Prints out the SH bank information, either the “tree” structure for an event or a list of all SH banks.
Source Code: Deck `SHTREE, SHLIST`, Patch `SIMULATION`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                  ======
SUBROUTINE SHTREE(ILUNIT)

                  ======
SUBROUTINE SHLIST(ILUNIT)
```

Calling Arguments:

Input argument
`INTEGER` `ILUNIT` Logical unit number for printout

Usage Notes: `SHTREE` is limited to printing out 15 generations in the tree.
Person Responsible: M. Bates

13.3 DELNAM

Function: Converts the Delphi convention particle codes to a name.
Source Code: Deck `DELNAM`, Patch `SIMULATION`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                      ======
SUBROUTINE DELNAM(IDELPH,CNAME)
                             =====
```

Calling Arguments:

Input argument

`INTEGER`	`IDELPH`	Delphi simulation particle code

Output argument

`CHARACTER*4`	`CNAME`	Particle name

Usage Notes:
Person Responsible: M. Bates, W. Murray

13.4 SPATKR, SPAPER

Function: Gets TKR or perigee parameters for a simulation truth particle.
Source Code: Deck SPATKR, SPAPER, Patch SIMULATION, Cards DSTANA, Library DSTANA, Disk PHY.
Calling Sequence:

```
                          ======
SUBROUTINE SPATKR(LSIMPA,SIMTKR,SIMTKZ)
                                    ====== ======

                          ======
SUBROUTINE SPAPER(LSIMPA,SIMPER,SIMERR)
                                    ====== ======
```

Calling Arguments:

Input argument

INTEGER	LSIMPA	Link to simulation PA bank

Output arguments

REAL	SIMTKR(6)	Radial TKR parameters at creation point $(R, R\Phi, z, \theta, \phi, -Q/p)$
REAL	SIMTKR(6)	Planar TKR parameters at creation point $(x, y, z, \theta, \phi, -Q/p)$
REAL	SIMPER(5)	Perigee parameters $(\epsilon, z, \theta, \phi, \kappa)$
REAL	SIMERR(5)	Perigee parameter errors

Usage Notes: The "error matrix" returned by SPAPER is usually zero. However, if the particle was created outside a material surface, such that the extrapolation to the perigee crosses a surface, then in principle, the perigee is uncertain to some degree. In these cases, the error matrix returned is actually negative and is only meaningful when added to an existing matrix. For example, a χ^2 comparison of the measured and truth perigee parameters should include this error matrix along with the measured errors. This is never relevant for SPATKR as the reference surface for the TKR is takes as the creation point, so no surfaces are ever crossed.
Person Responsible: P. Dauncey

13.5 TKREAS

Function: Associates reconstructed and truth tracks in simulation events.
Source Code: Deck TKREAS, Patch TKREAS, Cards TKREAS, Library DSTANA, Disk DSTFIX.
Calling Sequence:

```
                  ===== ====
SUBROUTINE TKREAS(IFLAG,BMAG)
```

Calling Arguments:

Input arguments

INTEGER	IFLAG	Flag controlling association (see below)
REAL	BMAG	Magnetic field in Tesla

Usage Notes: The values of IFLAG are 10*K+N, where K should be zero (no other options yet exist) and N controls how the links are changed. N=1 resets all links to zero and then only links tracks which have successfully been matched. N=2 sets all links for matched tracks and keeps the Delana link for those where no match is found. N=3 only attempts to find links for those tracks for which Delana could not find a link. The first option is recommended, so the normal use has IFLAG=1.
This routine is called after DSTFIX when creating the Short-DST.
Person Responsible: T. Todorov.

14 DST Reduction and MiniDST Routines

Routines used for Superteam 3 DST size reduction.
They allow to modify the three main parts of the DST: The Pilot record, the PA banks in the reconstructed PV bank part, the links to the TOP bank. The modifications are speeded up by copying all wanted informations in intermediate arrays and by changing only once the Pilot record or the PA bank. Nevertheless routines to modify only a few times per event the DST, exists. For the Pilot record, one can drop or add blocklet using BLCKDRP and BLCKCRE, and for the DST PA bank, one can add or drop an extra module using EXMCRE and EXMDRP.
Here follow an example of modifications:

1. **Pilot record**

```
C-- Want to keep only blocklet 0 8 and 14 from the Pilot and
C-- add blocklet 15 (sub-id 10):
      CALL BLCKINI
C-- Copy Blocklets 0 ,8 ,14
      CALL BLCKCOP(0,0,IER)
      CALL BLCKCOP(8,0,IER)
      CALL BLCKCOP(14,0,IER)
C-- Add a blocklet 15,10 from MYTAG(21) an array from my analysis
      CALL BLCKADD(15,10,21,MYTAG,IER)
C-- Update the Pilot
      CALL BLCKUPD(IER)
```

2. **Extra modules**

```
C-- Copy xtra 0,2,3,4 and add DEDX information on xtra 19
C-- Be carefull your track indexes must be decalred through MZLINK
      COMMON/MYLINK/LWPA,LWPV
      LOGICAL FIRSTPASS/.TRUE./
      IF (FIRSTPASS) CALL MZLINK(IUXDST,'/MYLINK/',LWPA,LWPV,LWPA)
      FIRSTPASS=.FALSE.
C-- Loop on PV and PA banks
      LWPV = LQ(LDTOP-1)
      DO 10 WHILE (LWPV.NE.0)
        LWPA = LQ(LWPV-1)
        DO WHILE (LWPA.NE.0)
C-- Copy xtra module 0,2,3,4
          CALL EXMINI
          CALL EXMCOP(LWPA,0,IER)
          CALL EXMCOP(LWPA,2,IER)
          CALL EXMCOP(LWPA,3,IER)
          CALL EXMCOP(LWPA,4,IER)
C-- Create a DEDX results xtra module 19
          CALL  GETDEDX(LWPA,DEDX(1),DEDX(2),DEDX(3),DEDX(4)
          CALL EXMADD(19,4,DEDX,IER)
```

```
C-- Update the PA bank
          CALL EXMUPD(LWPA,IER)
C-- end of loops
   20   LWPA = LQ(LWPA)
   10 LWPV = LQ(LWPV)
```

3. **TOP bank links**

```
C-- Drop link 15 and add a 17 link to put DJET Pilot record in
      CALL LINKDRP(15,IER)
      CALL LINKCRE(17,IER)
      LDJET = KBLCKAD(10,0)
      NWDJET = IUPILT(LDJET+1)
      CALL MZBOOK(IUXDST,LJET,LDTOP,-17,'DJET',0,0,NWDJET,2,0)
C-- It's an integer bank fill it with ICONT(1:NDATA)
      CALL UCOPY(IUPILT(LDJET+1),IQ(LJET+1),NWDJET)
```

14.1 BLCKINI

Function: Initialise BLCKCOM array in MODDST package.
Source Code: Deck `BLCKINI`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE BLCKINI
```

Calling Arguments: None
Usage Notes: This routine initialise the IPIL array in BLCKCOM common before any Pilot Record blocklet copy. It must be called once per event, before any over routine calls of MDBLCK package.
Person Responsible: L. Mirabito

14.2 BLCKCOP

Function: Copy the Pilot record blocklet identified by its identifier ID and its subidentifier ISUB in the IPIL array of BLCKCOM common.
Source Code: Deck `BLCKCOP`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                        == ====
SUBROUTINE BLCKCOP(ID,ISUB,IER)
                             ===
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Identifier of the blocklet to be copied
`INTEGER`	`ISUB`	Subidentifier of the blocklet to be copied

Output argument

`INTEGER`	`IER`	0 Copy has been done
		1 Inconsistent Blocklet length found
		2 No Blocklet with such identifiers found

Usage Notes: BLCKINI routine must be called once in the event before the first call to BLCKCOP.
Person Responsible: L. Mirabito

14.3 BLCKADD

Function: Add an extra Pilot record blocklet in IPIL array of BLCKCOM common.
Source Code: Deck `BLCKADD`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                   == ==== === =====
SUBROUTINE BLCKADD(ID,ISUB,LEN,ICONT,IER)
                                     ===
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Identifier of the blocklet to be added
`INTEGER`	`ISUB`	Subidentifier of the blocklet to be added
`INTEGER`	`LEN`	Length of the information in the ICONT array
`INTEGER`	`ICONT`	Array containing information to be stored as blocklet ICONT(1:LEN)

Output argument

`INTEGER`	`IER`	0 Blocklet create in IPIL array 1 Not enough space to add it 2 Inconsistant blocklet length in IPIL 3 Already existing Blocklet

Usage Notes: BLCKINI routine must be called once in the event before the first call to BLCKADD.
Person Responsible: L. Mirabito

14.4 BLCKUPD

Function: Update the content of IUPILT array in PXCHDR with the content of IPIL array of BLCK-COM common.
Source Code: Deck `BLCKUPD`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
SUBROUTINE BLCKUPD(IER)
                   ===
```

Calling Arguments:

Output argument

`INTEGER IER`	0 Copy is done 1 Insufficient length in IUPILT

Usage Notes: In case of insufficient length, the copy is not done.
Person Responsible: L. Mirabito

14.5 BLCKCRE

Function: Add an extra Pilot record blocklet in IUPILT array of PXCHDR common.
Source Code: Deck `BLCKCRE`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                   == ==== === =====
SUBROUTINE BLCKCRE(ID,ISUB,LEN,ICONT,IER)
                                     ===
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Identifier of the blocklet to be added
`INTEGER`	`ISUB`	Subidentifier of the blocklet to be added
`INTEGER`	`LEN`	Length of the information in the ICONT array
`INTEGER`	`ICONT`	Array containing information to be stored as blocklet ICONT(1:LEN)

Output argument

`INTEGER`	`IER`	0 Blocklet create in IUPILT array 1 Not enough space to add it 2 Inconsistant blocklet length in IUPILT 3 Already existing Blocklet

Usage Notes: It's a standalone routine. If only one blocklet is added it is as fast as calls to BLCK-INI,BLCKADD and BLCKUPD.
Person Responsible: L. Mirabito

14.6 BLCKDRP

Function: Drop a Pilot record blocklet in IUPILT array of PXCHDR common.
Source Code: Deck `BLCKDRP`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                      == ====
SUBROUTINE BLCKDRP(ID,ISUB,IER)
                           ===
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Identifier of the blocklet to be dropped
`INTEGER`	`ISUB`	Subidentifier of the blocklet to be dropped

Output argument

`INTEGER`	`IER`	0 Blocklet deleted in IUPILT array 1 Inconsistant blocklet length in IUPILT

Usage Notes: It's a standalone routine, no call to BLCKINI is needed.
Person Responsible: L. Mirabito

14.7 BLCKCUR

Function: Cure unreadable Pilot record by making a copy of all blocklets until length is too big or a corrupted blocklet is found.
Source Code: Deck `BLCKCUR`, Patch `MDBLCK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
SUBROUTINE BLCKCUR(NBOUT,IER)
                   ===== ===
```

Calling Arguments:

Output arguments

`INTEGER`	`NBOUT`	Number of blocklets copied
`INTEGER`	`IER`	0 No corruption found in IUPILT array
		1 Inconsistant blocklet length in IUPILT
		2 Length is too big in IUPILT

Usage Notes: It's a standalone routine, no call to BLCKINI is needed.
Person Responsible: L. Mirabito

14.8 EXMINI

Function: Initialise EXMCOM array in MODDST package.
Source Code: Deck `EXMINI`, Patch `MDEXM`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
SUBROUTINE EXMINI
```

Calling Arguments: None
Usage Notes: This routine initialise the EXM array in EXMCOM common before any extra module copy. It must be called once per track, before any over routine calls of MDEXM package.
Person Responsible: L. Mirabito

14.9 EXMCOP

Function: Copy the extra module identified by its identifier ID in the EXM array of EXMCOM common.
Source Code: Deck `EXMCOP`, Patch `MDEXM`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                  === ==
SUBROUTINE EXMCOP(LPA,ID,IER)
                         ===
```

Calling Arguments:

Input arguments

`INTEGER LPA`	Address of the PA bank
`INTEGER ID`	Identifier of the extra module to be copied

Output argument

`INTEGER IER`	0 Copy has been done
	1 No extra module with such identifier found

Usage Notes: EXMINI routine must be called once in the track before the first call to EXMCOP.
Person Responsible: L. Mirabito

14.10 EXMADD

Function: Add an extra module in EXM array of EXMCOM common.
Source Code: Deck `EXMADD`, Patch `MDEXM`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                  == === =====
SUBROUTINE EXMADD(ID,LEN,ICONT,IER)
                               ===
```

Calling Arguments:

Input arguments

`INTEGER`	`ID`	Identifier of the extra module to be added
`INTEGER`	`LEN`	Length of the information in the ICONT array
`INTEGER`	`ICONT`	Array containing information to be stored as extra module ICONT(1:LEN)

Output argument

`INTEGER`	`IER`	0 extra module create in EXM array 1 Too many extra module (>50) 2 New length is too big (>10000)

Usage Notes: EXMINI routine must be called once in the track before the first call to EXMADD.
Person Responsible: L. Mirabito

14.11 EXMUPD

Function: Update the content of PA bank with the content of EXM array of EXMCOM common.
Source Code: Deck `EXMUPD`, Patch `MDEXM`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                          ===
SUBROUTINE EXMUPD(LPA,IER)
                                ===
```

Calling Arguments:

Input argument

`INTEGER LPA`	Address of the PA bank

Output argument

`INTEGER IER`	0 Copy is done
	$\neq 0$ IQUEST(1) from MZPUSH

Usage Notes: LPA must be declared in a common through MZLINK, if any garbage collection occured. Copy is done only if MZPUSH succeeded.
Person Responsible: L. Mirabito

14.12 EXMCRE

Function: Add an extra module in PA bank.
Source Code: Deck `EXMCRE`, Patch `MDEXM`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                  === == === =====
SUBROUTINE EXMCRE(LPA,ID,LEN,ICONT,IER)
                                   ===
```

Calling Arguments:

Input arguments

`INTEGER`	`LPA`	Address of the PA bank
`INTEGER`	`ID`	Identifier of the extra module to be added
`INTEGER`	`LEN`	Length of the information in the ICONT array
`INTEGER`	`ICONT`	Array containing information to be stored as extra module ICONT(1:LEN)

Output argument

`INTEGER`	`IER`	0 extra module create in PA bank $\neq$ 0 IQUEST(1) from MZPUSH

Usage Notes: It's a standalone routine. If only one extra module is added it is as fast as calls to EXMINI,EXMADD and EXMUPD. The speed of the modification procedure is directly related to the number of calls to MZPUSH, consequently, this routine and EXMDRP must not be called more than once per track. In all other cases it is more powerfull to use EXMINI, EXMADD, EXMCOP and EXMUPD.
Person Responsible: L. Mirabito

14.13 EXMDRP

Function: Drop an extra module from the PA bank.
Source Code: Deck `EXMDRP`, Patch `MDEXM`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                   === ==
SUBROUTINE EXMDRP(LPA,ID,IER)
                          ===
```

Calling Arguments:

Input arguments

`INTEGER`	`LPA`	Address of the PA bank
`INTEGER`	`ID`	Identifier of the extra module to be dropped

Output argument

`INTEGER`	`IER`	0 extra module deleted in PA bank
		$\neq 0$ IQUEST(1) from MZPUSH

Usage Notes: It's a standalone routine, no call to EXMINI is needed.
Person Responsible: L. Mirabito

14.14 LINKDRP

Function: Clear a link from the TOP bank.
Source Code: Deck `LINKDRP`, Patch `MDLINK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                       ===
SUBROUTINE LINKDRP(LNK,IER)
                            ===
```

Calling Arguments:

Input argument

`INTEGER LNK`	Link of the TOP bank to clear

Output argument

`INTEGER IER`	0 The link is cleared 1 No link exist $\neq 0,1$ Zebra code error coded

Usage Notes: It drops the bank linked to LNK if it exist. The link always exists but points to 0.
Person Responsible: L. Mirabito

14.15 LINKCRE

Function: Add a link from the TOP bank.
Source Code: Deck `LINKCRE`, Patch `MDLINK`, Cards `MODDST`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                     ===
SUBROUTINE LINKCRE(LNK,IER)
                         ===
```

Calling Arguments:

Input argument

`INTEGER`	`LNK`	Link of the TOP bank to clear

Output argument

`INTEGER`	`IER`	0 The link is added 1 The link already exists $\neq$ 0,1 Zebra code error coded

Usage Notes: It adds a link to the TOP bank, but no bank is linked to it. To link properly a data set to this link you have to make the following calls:

```
C-- Create link 18
      CALL LINKCRE(18,IER)
      IF (IER.NE.0) RETURN
C-- Create a bank at address LDATA pointed by LQ(LDTOP-18)
      CALL MZBOOK(IUXDST,LDATA,LDTOP,-18,'DATA',0,0,NDATA,2,0)
C-- It's an integer bank fill it with ICONT(1:NDATA)
      CALL UCOPY(ICONT(1),IQ(LDATA+1),NDATA)
      ....
```

Person Responsible: L. Mirabito

15 Useful Utility Routines

Various useful routines for bits and pieces.

15.1 DSTQID, DSTQIE

Function: Finds “type” of data for processing-specific applications
Source Code: Deck DSTQID, Patch DSTIO, Cards DSTANA, Library DSTANA, Disk DSTANA.
Calling Sequence:

```
SUBROUTINE DSTQID(CHTYPE)
                  ======

               ======
ENTRY DSTQIE(LPRINT)
```

Calling Arguments:

Input argument

INTEGER	LPRINT	LUN for printout

Output argument

CHARACTER*4	CHTYPE	type of data for current event

Usage Notes: CHTYPE is a 4-character variable. The first two characters specify the year; that is CHTYPE(1:2) is '90','91' etc. The third speficies the processing letter; that is CHTYPE(3:3) is 'E', for Delana-E etc. The fourth character, CHTYPE(4:4), specifies the Short-DST production; ' ' for standard DST, '1' for the first processing etc.
The routine DSTQID works for both data and production MC and gets the PX version number firstly from the /UXCEVT/ common varible NVERDS; if this is zero, then it tries to unpack it from the DST, using LDTOP.
DSTQIE can be called at the end of a job to print out a table of statistics of types of events found in calls to DSTQID. Note, this counts numbers of calls with different event numbers to DSTQID, not calls directly, so multiple calls to DSTQID in an event will only give a single count in the statistics table. DSTQIE is called from the Delphi standard end routine DSTEND (3.1).
Person Responsible: P. Dauncey

15.2 TKSTART

Function: Extract starting points
Source Code: Deck `TKSTART`, Patch `TKSTART`, Cards `DSTANA`, Library `DSTANA`, Disk `DSTANA`.
Calling Sequence:

```
                          ===
SUBROUTINE TKSTART(LPA,RSTART,R_TPC,R_ID)
                               ====== ===== ====
```

Calling Arguments:

Input argument

`INTEGER`	`LPA`	LPA pointers for tracks

Output arguments

`REAL`	`RSTART`	TK starting point
`REAL`	`R_TPC`	TPC starting point
`REAL`	`R_ID`	ID Jet Chamber starting point

Usage Notes:

Person Responsible: M.Caccia, E.Torassa

15.3 BPILOT

Function: Gets the magnetic field value from the pilot record and fills various commons.
Source Code: Deck BPILOT, Patch UTILITY, Cards DSTANA, Library DSTANA, Disk PHY.
Calling Sequence:

```
SUBROUTINE BPILOT(BTESLA,BGEVCM)
                   ====== ======
```

Calling Arguments:

Output arguments

REAL	BTESLA	Magnetic field in Tesla
REAL	BGEVCM	Magnetic field in GeV/cm

Usage Notes: This routine also fills several commons with the magnetic field, which are used by other routines (track fitting and extrapolations). These are /PXCONS/ (section 2.6), variable BMAG, which is the field in Tesla, /EXXFLD/, variable CONSB, which is the field in GeV/cm, /FKFILD/, double precision variable CONSB, which is the field in GeV/cm and /UFIEL3/, variable SCALEC, which is the scale factor for the field relative to the nominal field of 1.2 Tesla. This latter variable allows UFIELD to be used.
Note, the relationship between the magnetic field in Tesla and in GeV/cm is

$$B(GeV/cm) = \texttt{CONSA} \times B(Tesla)$$

where CONSA is the constant in /PXCONS/.
BPILOT is called for every event by the standard Delphi routine DSTEVT (section 3.1).
Person Responsible: P. Dauncey

15.4 EPILOT

Function: Gets the LEP energy value from the pilot record and fills the `/PXCONS/` common.
Source Code: Deck `EPILOT`, Patch `UTILITY`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
SUBROUTINE EPILOT(ECMASS)
                  ======
```

Calling Arguments:

Output argument

`REAL`	`ECMASS`	LEP centre-of-mass energy in GeV

Usage Notes: This routine also fills `/PXCONS/` (section 2.6), variable `EBEAM`, which is the beam energy in GeV.
`EPILOT` is called for every event by the standard Delphi routine `DSTEVT` (section 3.1).
Person Responsible: P. Dauncey

15.5 ZEBPRT

Function: Prints out a whole Zebra bank in an easy-to-read format.
Source Code: Decks `ZEBPRT`, Patch `UTILITY`, Cards `DSTANA`, Library `DSTANA`, Disk `DSTFIX`.
Calling Sequence:

```
                  ====== ======
SUBROUTINE ZEBPRT(LZBANK,LUNPRT)
```

Calling Arguments:

Input arguments

`INTEGER`	`LZBANK`	Link to required bank
`INTEGER`	`LUNPRT`	LUN for printout

Usage Notes: For protection against huge amounts of printout, or being called with a invalid link, the number of data words printed is limited to 1000.
Person Responsible: P. Dauncey

15.6 UREAL, UDBLE

Function: Converts arrays from `REAL` to `DOUBLE PRECISION` and vice versa.
Source Code: Decks `UREAL, UDBLE`, Patch `UTILITY`, Cards `DSTANA`, Library `DSTANA`.
Calling Sequence:

```
                  ======         ======
SUBROUTINE UREAL(DARRAY,RARRAY,NARRAY)
                         ======
                  ======         ======
SUBROUTINE UDBLE(RARRAY,DARRAY,NARRAY)
                         ======
```

Calling Arguments:

Input arguments

	`xARRAY(*)`	Array to be converted
`INTEGER`	`NARRAY`	Size of array

Output argument

	`xARRAY(*)`	Converted array

Usage Notes: `UREAL` converts a `DOUBLE PRECISION` array to `REAL`. `UDBLE` converts a `REAL` array to `DOUBLE PRECISION`.
Person Responsible: P. Dauncey

15.7 ATN2DL, DATN2L, DATG

Function: Calculates arctangent in DELPHI convention; $0 < \phi < 2\pi$.
Source Code: Decks `ATN2DL, DATN2L, DATG`, Patch `UTILITY`, Cards `DSTANA`, Library `DSTANA`, Disk `PHY`.
Calling Sequence:

```
                            = =
REAL FUNCTION ATN2DL(X,Y)
              ======
```

```
                                        = =
DOUBLE PRECISION FUNCTION DATN2L(X,Y)
                          ======
```

```
                                      = =
DOUBLE PRECISION FUNCTION DATG(X,Y)
                          ====
```

Calling Arguments:

Input arguments

`REAL, D.P.`	`X`	x component of vector for arctangent
`REAL, D.P.`	`Y`	y component of vector for arctangent

Outputs

`REAL`	`ATN2DL`	value of arctangent
`DOUBLE PRECISION`	`DATN2L`	value of arctangent
`DOUBLE PRECISION`	`DATG`	value of arctangent

Usage Notes: `ATN2DL` has `REAL` arguments and a `REAL` result. For a `DOUBLE PRECISION` version, use `DATN2L`, which has `DOUBLE PRECISION` arguments and a `DOUBLE PRECISION` result. `DATG` is the same as `DATN2L` as is supplied to be compatible with the CERNLIB `REAL` function `ATG`.
Person Responsible: P. Dauncey

Index