January 14, 2003

Nomad Reconstruction Software

Nomad DST Package

Version v7r4

Kevin Varvell[1]

Abstract

This note describes the Data Summary Tape (DST) package for NOMAD. The version described here is designed for production and examination of DSTs using recon version 7r7 and related libraries.
This package includes :

- Code to produce the DST ZEBRA structure from phase2 banks
- Additional temporary code to add information to the DST structure without a repass of phase2.
- Access routines to the DST structure for C and Fortran programmers, including the package of Fergus Wilson.

[1] Falkiner High Energy Physics Department
The University of Sydney
(KEV@PHYSICS.USYD.EDU.AU)

Table of Contents

List of Figures

Chapter 1: Introduction

This note describes the ZEBRA-based Data Summary Tape (DST) structure for the NOMAD offline. The need for a relatively compact representation of the output of the matching process (reconstruction phase 2), which can form a common starting point for analysis, was one of the main motivations for the setting up of such a structure.

1.1 A little history

The DST structure is based exclusively on the ZEBRA bank structure available after Recon phase 2 processing has taken place. The first release was given the name v1r0, and was used to produce DSTs from the results of recon prod4 production on 1995 data. That release was *only* intended to be used with prod4 output tapes from 1995. It is unsuitable for use with recon v7 compatible libraries in two respects - it has code to fix bugs in prod4 output which have subsequently been fixed, and it requires some extra "non-standard" code in order to access information not available on prod4 output but which is now available with recon version 7 production.

An updated version for prod4 output, v1r1, was subsequently released. This fixed some bugs in v1r0, and included in the DST library itself the access package of Fergus Wilson [2]. DSTs were not produced with this version.

Version v7r1 was produced in March 1997. This was intended for use with the first test of production made with recon v7 and the associated libraries at the time. Much feedback from its use was obtained which was incorporated into release v7r2 [3], intended for use with reconv7r7, phase2v7r3c and related libraries. This release was made in June 1997.

Version v7r3, released in February 1998, incorporated several new features. It was set up to run on the *same* phase2 output that was used for v7r2, in order to avoid requiring a time consuming new pass of phase2. This meant that some of the new features in v7r3 had to be be added *ad hoc* at the DST level, which was messy but expedient.

The present release, v7r4, is a minor iteration on v7r3. Two updates, due to B. Yabsley, have been added. These correct the TRD shared hit handling which was not handled correctly in the production to produce v7r3, and add a single word to allow goodness of fit of V0 vertices to the primary to be ascertained. The ad hoc code which eliminates the need for a full phase2 reprocessing is present in this version also, as the likelihood of a further phase2 processing of the NOMAD data decreases with time.

1.2 Layout of this document

A brief overview of the package is given in chapter 2. The structure in its present form will be described in chapter 3. In chapter 4, the access routines which are available for extracting information from the structure will be described, and some examples given for C and Fortran. A more detailed description of the banks is given in chapter 5. In chapter 6, some details of the organisation of the DST code itself will be given, intended for specialists who may need to interact with the source code. Finally, in chapter 7, release notes for the current version are given and some relevant issues discussed.

A Web Page exists which gives up to date information on the current status of DST development. This may be accessed from the NOMAD Web Pages by going to the FARINE page and following the "dst" link from there [1].

This document may be found in the file $NOMAD˙PS/dstv7r4.ps on the NOMAD cluster. The corresponding bank documentation can be found in the file $NOMAD˙PS/dstv7r4˙bankdoc.ps. As indicated

above, this version of the library includes the access code package developed by Fergus Wilson [2], which will be referred to in this document as the "dstaccess" package. At the time of writing, the documentation for "dstaccess" can be found in the file $NOMAD˙PS/dstgenv1r2.ps, but it is wise to check the above web page for the up-to-date location.

1.3 Acknowledgements

Because of the fact that this version of the DST package is designed to be run on phase2 output that has not been reprocessed through an updated phase2 package, code has had to be provided to obtain or calculate some of the new information that is not provided in the present phase2 output ZEBRA banks, and to correct some of the problems with the phase2 from which DST versions v7r2 and v7r3 were produced.

This code has come from various sources. Here is a hopefully complete list of people providing this code - I apologise if anyone has been omitted: Dario Autiero, Antonio Bueno, Marco Contalbrigo, Luigi DiLella, Achim Geiser, Peter Hurst, Stefano Lacaprara, Domizia Orestano, Fabrizio Salvatore, Bruce Yabsley.

Chapter 2: Brief Description of the Package

The package in its present form has been designed to be interfaced to recon, although the interface is fairly clean. In principle, therefore, it can be called from other executives. For example, it could be called from camel to produce a DST structure from a recon output file which has previously had phase 1 and 2 processing performed.
The DST package has been written primarily in FORTRAN, but contains some C code as well, especially when coding the access functions. The routines described in this chapter are FORTRAN routines; however, many routines written in one language mave been rendered callable from the other through the cfortran package - one example is given below.

2.1 Initialisation

An initialization routine is provided with the name **DSTINI**, which currently specifies which match combinations of subdetectors have been defined in this package, and sets their identifiers. It also initializes variables used to compile statistics on the DST size. This routine is called during the initialization phase. If not called explicitly by the executive program, it will be called the first time a call to **CreateDST** or **PrintDST** is invoked. These are introduced below.

2.2 End of run

An end of run routine, **DSTEND** is also provided. At present this routine prints some statistics on the size of the DST, provided that the DST print level is set to 1 or greater.

2.3 Creating a DST

For each event, the DST ZEBRA structure is created from the full ZEBRA structure by a call to the subroutine **CreateDST()**.
In this version of the package, **CreateDST()** also calls a preprocessing routine, **DstPreProcess()**, which performs the necessary additional work required because phase2 has not been reprocessed prior to input the DST package. This will be described in more detail in chapters 6 and 7.

2.4 Print output

A flag to control the amount of printout produced is used by the package. To set this flag, the routine **dstSETPRFLAG(ILEVEL)** should be called, where the higher the value of **ILEVEL**, the more printout is obtained. If set to 0, no printout occurs, while if set to 1, summary information is printed if **DSTEND** is called.
A routine to print the contents of the DST bank structure has been provided with the package, which can be invoked with a call to **PrintDST()**. In version 1 of the package this routine provided an output in a wide format (up to 132 characters), which required special handling when viewing or printing the output. From version v7r1 onwards, a maximum width of 80 characters is respected in the output, and the word names used are identical to those in the bankdoc.
The routines mentioned in this chapter have been rendered C callable, and can be invoked by capitalising the subroutine name, e.g.

```
    PRINTDST();
```

In this case, the user should include the header file **dstgen.h** in the C source file.

2.5 Debug output

In a similar manner to the print flag described above, a debug flag is used by the package, primarily of use in development. In this case the routine **dstSETDBFLAG(ILEVEL)** should be called, with increasing **ILEVEL** giving increasing amounts of output (at present ILEVEL set to 2 gives maximum output).

2.6 Data cards in recon

Version 7 of recon introduced a data card DSTF which can be used to set the print and debug flags as desired. Two arguments are taken, the first setting the print level, the second the debug level. The default is a print level of 1 and a debug level of 0, as shown below.

```
* ---------------------------------
* -> DSTF
*    Print / Debug for DST
DSTF 1       0
* ---------------------------------
```

In order to get DST output from a recon run, two cards must be adjusted, the LUNS card and the TRIM card.
The following LUNS card will place the DST ZEBRA structures into fort.13 and the full phase2 output (including the DST ZEBRA structures) into fort.11.

```
* ---------------------------------
* -> LUNS
*    Input/Ouput/FzIn/FzOut/DSTOut
LUNS 1=5   2=6  3=10  4=11   5=13
* ---------------------------------
```

The following TRIM card will ensure that fort.13 contains only the DST structure.

```
* ---------------------------------
* -> TRIM
*    ----- if DSTout in LUNS  -----
*    Which structure to keep (DST)
*      1 : >0 --> Drop RAWD
*          >1 --> Drop SEVT/EVT/DETE
*      2 : =1 --> Drop DETE
*          =2 --> Drop DETE but BDCH
*      3 : =1 --> Compress TX/TXD
*          =2 --> Drop     TX/TXD
*
TRIM 1=2   2=1   3=2
* ---------------------------------
```

2.7 Histograms

Some code exists which will histogram all words stored in the DST structure. For more details refer to chapter 6.

Chapter 3: The DST Structure

The DST banks are placed in the overall ZEBRA structure at link -6 of the top level NOMA bank, which was reserved for such a structure from early in the experiment.

Figure 3.1 shows the layout of the banks. A header **DST** bank acts as a hanger to which the banks containing real information are attached. The following types of banks can be found in the structure:

- **EVS**: The **EV**ent **S**ummary bank contains some global summary information for the event, which may be useful for users to make some of their selections without looping through the detailed structure.
- **RVXS:** The **R**econstructed **V**erte**X** **S**ummary bank contains a list of the reconstructed vertices in the event.
- **MAS**: The **MA**tch **S**ummary banks are a linear structure containing one bank per line of the match table, or if you like, one bank per "object" that the matching has produced.
- **LEPS**: The **LEP**to **S**ummary bank contains a list of the particles in the initial interaction generated using LEPTO (in Neglib), as given in the LUJETS common.
- **SVXS**: The **S**imulated **V**erte**X** **S**ummary bank contains a list of the simulated vertices in the event.
- **STKS**: The **S**imulated **T**rac**K** **S**ummary bank contains a list of the simulated tracks (charged and neutral) contained in the event, as provided by the GENOM simulation.
- **SCAS**: The **S**imulated **CA**lorimeter **S**ummary bank contains a list of the calorimeter cells which have deposited energy in them, as provided by the GENOM simulation.

Note that some of the banks are only available for Monte Carlo events. These are the LEPS, SVXS, STKS and SCAS banks. The EVS, RVXS and MAS banks are available for both data and Monte Carlo.

The detailed contents of the banks just described can be found in the bank documentation, and also in chapter 5. A copy of the current working version of the DST bank document can be found in $NOMAD˙PS/dstv7r4˙bankdoc.ps on the cluster.

Some further information and discussion on the banks in the structure will be given in the following sections.

3.1 DST Header Bank

The only information stored in the header bank is the version number of the DST, stored as a decimal number. This can be retrieved using the C function (of type float, taking as argument the pointer to the DST bank) **DstVersion(˙Dst Dst)** or the FORTRAN function (of type REAL) **GetDstVersion()**.

3.2 Event Summary Bank

The purpose of this bank is to provide some global summary information for the event. It may be of use in making some very crude event selections without having to loop through the banks containing the more detailed information.

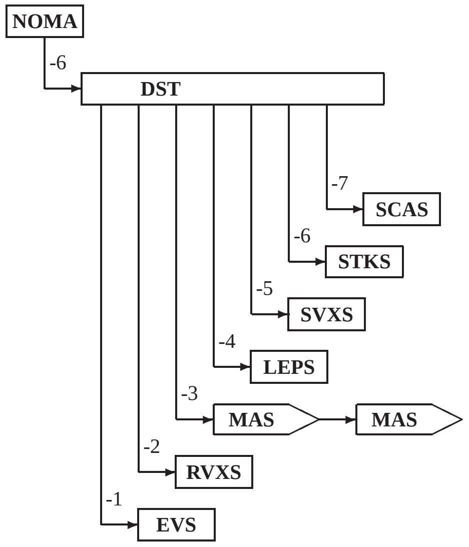

Figure 3.1: The DST bank structure

3.3 Reconstructed Vertex Summary Bank

This is essentially a list of vertices which have been reconstructed by the vertex package, and the bank is thus implemented as a word giving the number of vertices stored, followed by a series of fixed length blocklets one per vertex.

Note the following:

- End point vertices, which have vertex type 10 in the VTX banks of the full structure, have been suppressed on the DST if there are no neutral objects attached. Also skipped over are vertices of type 11 (dummy neutrals from the unassociated matched clusters - these were type 8 in earlier versions of the VTX structure) and vertices of type -1 (which are dummy primary vertices produced when no primary vertex is reconstructed, in order to have somewhere to hang the TRK banks).

3.4 Match Summary Banks

The format of the MAS banks requires some additional explanation which will hopefully help in understanding the bankdoc.

Each MAS bank corresponds to a line in the match table maintained by the matching engine, and can be thought of as an object which has resulted from the matching process. Every object in the match table is realised as a MAS bank, and so the size of the MAS structure (and amount of ZEBRA overhead) is determined by what objects the match algorithms employed by the matching engine choose to leave in the match table. This is an area where choices made by developers of matching algorithms can sensitively affect the ultimate size of the DST.

3.4.1 MAS bank structure

A MAS bank consists of a fixed set of header words followed by a number of subdetector blocklets. There is one header word per NOMAD subdetector, and this header word, if non-zero, indicates that the subdetector in question is contributing information to the match object. Moreover, if the subdetector information is present, the header word contains an offset from the start of the bank to the subdetector blocklet containing the information.

By storing information in this way, the match information can be made reasonably compact. For an overhead of one word per subdetector which must *always* be present, it is then only necessary to store those blocklets for which subdetector information is *actually* present in the match table, this information being reached within a particular MAS bank using the offset.

3.4.2 Subdetector blocklet structure and Identifiers

The first word of a subdetector blocklet is the blocklet type, indicating what sort of match this blocklet corresponds to. This is constructed in the form mmnn, where mm is the ID of the subdetector providing the information, and nn is the ID of the "seed" subdetector to which it is matched. The IDs range from 1 for drift chambers to 9 for HCAL, following the normal NOMAD convention. In addition, ID 10 has been reserved for the potential addition of the silicon prototype. IDs greater than 10 are used for special purposes. ID 11 is used for storing the alternative ECAL clustering of Gary Feldman, as will be explained later. ID 12 is used to store extrapolator information, and ID 13 to store Padova Bremsstrahlung strip information.

As examples of blocklet IDs, a drift chamber track is given ID 101, and an unassociated calorimeter cluster ID 404. If the calorimeter cluster is in fact matched to a drift chamber track, it would have ID 401. Subsequent words in the blocklet contain detailed information for the subdetector, and can be either integer or floating depending on the context. The bankdoc provides this information as IO:I (for integer) or IO:F (for floating).

This implementation based on a blocklet identifier again leads to compactness, since only the information relevant to a match to the specified seed subdetector need be stored, without storing empty words. It does make the bankdoc harder to read, however, and the point to note here is that for a given MAS bank, at most only *one* type of blocklet will be present for a given subdetector. For example, although the bankdoc describes for the calorimeter blocklet types 401 and 404, a MAS bank with a non-zero offset to the calorimeter in its header words will contain *either* a blocklet 401 (cluster matched to a drift chamber track) *or* a blocklet 404 (unassociated cluster), but not both.

Some of the MAS blocklets are variable in length, in cases where it is appropriate to store information for a number of objects at once. Examples are the blocklets 505 and 808, for muon chamber standalone tracks and events in FCAL respectively, and the blocklets 1101 and 1301, which store cross-reference lists of objects in the bremsstrahlung strips.

The structure of the MAS banks will probably become clearer after studying the examples of access given in chapter 4, or by looking at the output produced by the **PrintDST()** routine.

3.5 Lepto Summary Bank

This bank gives a summary of the event generation for the present event. A header gives the kinematic variables, including neutrino energy (and parent particle type), x_{Bj}, y_{Bj}, W^2, Q^2 and ν, along with the number of primary particles in the event. This header is followed by a series of blocklets, one per particle, giving the particle type, 4-momentum and mass of each particle produced in the fundamental interaction generated by LEPTO. Short-lived particles which will be subsequently decayed by GENOM are included. The kinematic information is derived from the LEPT bank and the particle list from the LUJT bank, the latter being essentially the information in the LUJETS common, minus the vertex information.

3.6 Simulated Vertex Summary Bank

This is a list of vertices which have been produced during the GENOM simulation of the event, using the information which gets stored in the SVTX banks. As in the case of the RVXS bank, the bank is implemented as a word giving the number of vertices stored, followed by a series of fixed length blocklets one per vertex. Because the number of vertices generated by a detailed GENOM simulation can be large, and we are primarily interested in vertices which may have been reconstructed using the vertex package, cuts will be developed to reduce the number of vertices stored. At present, end point vertices (type 10) are suppressed unless the track pointing to them is a muon, and a z cut at the centre of the HCAL is employed.

3.7 Simulated Track Summary Bank

The STKS bank is a list of tracks which have been produced during the GENOM simulation of the event, using the information which gets stored in the STRK banks. The bank is implemented as a word giving the number of tracks stored, followed by a series of fixed length blocklets one per track. Because the number of tracks generated by a detailed GENOM simulation can be large, and we are primarily interested in tracks which may have been reconstructed using the drift chamber code, cuts will be developed to reduce the number of tracks stored. At present the only cuts employed are to check that the beginning and end point vertices do not both lie beyond the preshower, and to require the track to have momentum greater than 30 MeV/c. Muon tracks are always retained. The number of tracks stored is still large following this selection, and many are short tracks caused by the way GEANT steps through the media of the NOMAD detector. More work is needed here to tune the track selection, especially as the size of the Monte Carlo DSTs is presently prohibitive.

The current implementation of the STKS bank contains some drawbacks which will be mentioned. One is that in choosing to store both charged and neutral "tracks" in the same bank, some waste of space is produced due to the necessity to have fixed length blocklets and to store different information in the two cases. Specifically, for charged tracks the position and 4-momentum at the first and last hits are stored (for consistency with what is stored for reconstructed tracks), whereas for neutrals this information is not present.

3.8 Simulated Calorimeter Summary Bank

The SCAS bank is a list of calorimeter cells which have seen energy deposition during the GENOM simulation of the event, using the information which gets stored in the RCAL bank. The bank is implemented as a word giving the number of cells stored, followed by a series of fixed length blocklets one per cell. Users should be aware that the spatial information stored in this bank is an offset relative to the centre of the relevant cell, as is the case in the RCAL bank.

The scheme for handling Monte Carlo calorimetric information is still rather rudimentary. Users are invited to suggest more appropriate information to be stored.

3.9 FZ Header

When the DST structure is output to a separate FZ file, the resulting ZEBRA structure has associated with it the same FZ header that the full phase1 and phase2 structure utilised. Thus the user has available the words stored there, such as the run and event numbers. These may be accessed transparently when reading a DST file using, for example, the /NDEVENT/ common block used in recon and camel, into which the header is unpacked upon reading in the event. The run and event number may be obtained in this way, for example, which is why these words do not appear explicitly in the DST bank structure.

Access functions are provided in the DST library for returning words commonly required from the FZ header. See the following chapter for more details.

3.10 Alternative ECAL clustering of Gary Feldman

The alternative ECAL clustering of Gary Feldman has been implemented in the following way. For convenience, these objects, which consist of clusters associated to a drift chamber track or an unassociated cluster just as in the case of the "official" clustering in the match engine, have been identified as belonging logically to a subdetector of name FEL and identifier 11. The 9 pointers to subdetector blocklets which appear at the top of the MAS bank have been extended to 13 (pointer 10 is reserved for the silicon prototype, and 12 and 13 are discussed in the next section). Following this pointer the user will arrive at blocklets with ID either 1101 (for cluster associated to a drift chamber track), 1104 (for a cluster which has been associated to a neutral HCAL cluster, *itself* associated to a standard ECAL cluster), 1109 (for a cluster associated to a standalone neutral HCAL cluster which is *not itself* associated to a standard ECAL

cluster, or 1111 (for an unassociated neutral cluster). In this way, the tools which are used to access information for the standard match objects can be used to access the alternative clusters in an analogous way.

If the drift chamber track to which a charged cluster is associated is actually an electron which has an associated bremsstrahlung strip, the cross-referencing information giving which objects form the brem strip is included in the 1101 blocklet, which becomes variable length.

Note that the use of the identifier 1104 is different between DST version v7r2 and version v7r3 and the present version. In v7r2, it referred to an unassociated, standalone Feldman ECAL cluster, while in the later versions identifier 1111 is used to better reflect its meaning. In the present version, 1104 refers to clusters which have been matched to those neutral HCAL clusters which are themselves matched to a standard ECAL cluster, as explained above. It is the introduction of matching results for Feldman ECAL clusters to neutral HCAL clusters which has necessitated this change, and users should be careful to take into account the change in meaning.

The make switch DST˙FELDMAN which was present in version v7r1 source code to build a library incorporating this alternative clustering is not used in later versions and can be omitted.

3.11 Extrapolator information

In version 1 of the DST library, extrapolator information at reference planes for PRS, CAL, HCAL and the two stations of the muon chambers was only stored if that particular subdetector matched to a drift chamber track. In the present library, extrapolator information is stored for all charged tracks if available, and appears as a MAS blocklet with identifier 1201 (using "subdetector" 12 to denote the extrapolator). The absence of this blocklet when blocklet 101 is present in an object indicates that no extrapolator information was stored for this track (strictly speaking that no TX/TXD structure was present in the phase2 output for this track).

3.12 Padova Bremsstrahlung Strip

In this version of the DST (from v7r2 onwards in fact) the Padova Bremsstrahlung Strip algorithm output is available for the first time. For those electron tracks which are found to have emitted bremsstrahlung, a blocklet of type 1301 is added to the MAS bank. This is an exact copy of the information in the DMBR bank which is produced by the algorithm at the phase 2 level. The detailed contents of the blocklet can be found in chapter 5. See also the note on bremsstrahlung algorithms below.

3.13 Notes on bank contents

3.13.1 Omissions

If words in the present DST are not filled, the most common reason for this is that the relevant words containing the information are not present in the current phase1 and phase2 ZEBRA structure produced by recon.

A more detailed description of what is missing can be found in Chapter 7.

Where words have not been filled, the following convention has been adopted:

- Integers and floating words which are necessarily positive by their context have been given the value integer -999 or floating -999.0.

- Integers and floating words which can have a sign have been given value integer 0 or floating 0.0.

Where this has been done at the DST level, it is indicated in the bank document (please report any omissions which are noticed, so that the bankdoc can be made more accurate). Note, however, that when a value is notionally available in the phase1 and phase2 ZEBRA structure, its value has been preserved in the DST. Therefore, if in fact it was not filled by the phase2 code, the convention used by that code must be ascertained in order to understand the DST contents.

3.13.2 Track lengths

With the present version of the DST, track lengths when present in the bankdoc are filled - this was not the case with releases earlier than v7r2.

3.13.3 Bremsstrahlung algorithms

Users should be aware that there are results of *three* types of bremsstrahlung algorithm stored on the DST. Information for the so-called “Padova” Bremsstrahlung algorithm will be found in MAS blocklets of ID 1301. Information for the algorithm of Gary Feldman will be found in MAS blocklets of ID 1101, with some cross-referencing information in MAS blocklets of ID 1111. Finally, some words relating to the algorithm which is present in the phase 2 calorimeter clustering code itself (specifically in **calgamma.c**) can still be found in the 404 MAS blocklet. These three sets of information should not be confused.

Chapter 4: Access Functions

Access functions to the DST ZEBRA structure are provided in order to ease its use for both C and FORTRAN programmers.

The advantage of using access routines is that any analysis code that the user writes is in principle immune to changes in the underlying ZEBRA structure. Where direct navigation of the structure is performed, changes to this structure must be reflected in changes to the user code, requiring both more work for the user and a heavier reliance on an up-to-date bank documentation (which of course should be available!).

For the above reasons, use of the access routines provided with the DST package is **highly recommended**. Suggestions for further access routines that should be provided can be made, although the number of further versions of the NOMAD DST library to be produced is probably rather limited at this point in the experiment.

Two packages of access code are available to the user, one developed along with the dst library, and a second, “dstaccess”, developed independently [2]. The examples which are given in this document refer to the former.

In the following discussion, some examples are given example numbers. The code for these examples can be found in the doc subdirectory of the dst source, enabling users to adopt them as templates for their own programs.

4.1 Access routines for C Programmers

4.1.1 Navigating the structure in C

The complete DST ZEBRA structure has been mapped onto corresponding C structures, in order to allow a C programmer to navigate the structure. The scheme used follows that outlined in the util documentation, using macros passed to the C preprocessor. For example, the DST source code contains, amongst others, the macros

```
REFERENCE_FROM_LINK(Dst, "DST ")
REFERENCE_FROM_REFERENCE(Dst, Mas, "MAS ", FirstMas)
REFERENCE_FROM_REFERENCE(Mas, Mas, "MAS ", Next)
```

which, upon generation of the library, makes available the functions

```
_Dst DstReferenceFromLink(int Ldst)
_Mas DstFirstMasReference(_Dst Dst)
_Mas MasNextReference(_Mas Mas)
```

In addition, some top entry access has been provided to the DST banks, in the form of functions **DstReference**, to provide a pointer, and the integer function **DstLink**, to provide a link to the ZEBRA bank.

Each C structure representation of a bank has an associated pointer type. As can be seen from the functions above, the pointer to the DST bank structure has type *˙Dst*, to a MAS bank structure *˙Mas*, and so on. In order to gain access to these pointer types, the include file **dstbanks.h** should be incorporated into your C source file. The include file **dstgen.h** prototypes the navigation routines mentioned above, and in fact includes **dstbanks.h**.

Navigation through the banks with C can be illustrated with some examples.

Suppose one wants to loop through the MAS banks. This could be achieved with the following code fragment:

```
#include "dstbanks.h"
#include "dstgen.h"

   _Dst  Dst;
   _Mas  Mas;

   Dst = DstReference();
   for (Mas = DstFirstMasReference(Dst); Mas; Mas = MasNextReference(Mas))
   {
      <Do something with this MAS bank>
   }
```

Having obtained the pointer to a particular MAS bank (that is, a particular object, or line, in the match table), it is then necessary to access the subdetector blocklets contained in this bank. An integer function **MasBlocklet** is provided to test whether a blocklet of a particular type is available.

Example 1 If the information desired comes from a muon matched to a drift chamber track (blocklet identifier 501), the above code fragment could become

```
#include "dstbanks.h"
#include "dstgen.h"

void example1(void)
{
   _Dst  Dst;
   _Mas  Mas;

   printf(" Calling EXAMPLE1 ...\n");

   Dst = DstReference();
   for (Mas = DstFirstMasReference(Dst); Mas; Mas = MasNextReference(Mas))
   {
      if (MasBlocklet(Mas, 501))
      {
         printf("Found blocklet 501. Do something with it.\n");
      }
   }
}
```

As can be seen, the function **MasBlocklet** returns a non-zero value when the specified blocklet exists for this object, and zero otherwise.

4.1.2 Low-level Access to Bank Contents in C

For those users who desire access at the individual word level to the C structure bank representation, appropriate access routines are provided. At the risk of unwieldly names, these have been implemented

using a naming scheme which can be deduced from knowledge of the bank name and the variable name within the structure. The variable names are those given in the ZEBRA bank documentation for the DST, and these names are case sensitive.

Examples will best illustrate the idea. Suppose one wishes to obtain the number of reconstructed vertices stored in the RVXS bank. From the bank doc, the listed variable name is **NVert**, and is an integer. By combining the bank name, with the first letter capitalised, **Rvxs**, with the variable name, we get the access function name **RvxsNVert**. Thus one might write

```
#include "dstbanks.h"
#include "dstgen.h"

   _Dst     Dst;
   _Rvxs    Rvxs;
   int      nvert;

   Dst   = DstReference();
   Rvxs  = DstRvxsReference(Dst);
   nvert = RvxsNVert(Rvxs);
```

Example 2 Suppose one wishes to loop through these vertices and print out the z position. To do this the access routine takes as an additional argument the “index” of the desired vertex (i.e. an integer in the range 1 to the number of vertices) within the bank. The code

```
#include "dstbanks.h"
#include "dstgen.h"

void example2(void)
{
   _Dst     Dst;
   _Rvxs    Rvxs;
   int      nvert, i;
   float    z;

   printf(" Calling EXAMPLE2 ...\n");

   Dst   = DstReference();
   Rvxs  = DstRvxsReference(Dst);
   nvert = RvxsNVert(Rvxs);
   if (nvert)
   {
      for (i = 1; i <= nvert; ++i)
      {
         z = RvxsZ(Rvxs, i);
         printf("z position of vertex is %f\n", z);
      }
   }
}
```

should do the job.

Names of access functions to the information in the blocklets of the MAS banks are constructed along similar lines, except that the names of the subdetectors which pertain to a given blocklet are included in the function name. As an example, functions accessing a blocklet with ID 101 (drift chamber track information), have names **MasDchDchxxxxx** where **xxxxx** is the variable name as given in the bank doc. For blocklet ID 501 (information from a muon to drift chamber match) the name would be **MasMuoDchxxxxx**. It can be seen that the rule is to replace the "subdetector" numerical ID, in the range 1 to 13, with the three letter character identifier, drawn from the set {**Dch, Trd, Prs, Cal, Muo, Sci, Vet, Fca, Hca, (Sit), Fel, Ext, Bre**}.

4.1.3 Higher-level Access to Bank Contents in C

Since it may be inefficient in many cases to call access functions that return a single word from the DST structure, there is a case for providing higher level access routines to return blocks of information. This functionality is provided through the "dstaccess" package, which is provided as part of the present library. For further information, the reader is referred to the appropriate documentation [2]. The examples mentioned in that document may also be found in the source code for the present library, in the directory /nomad/src/dst/v7r4/doc. The files have been renamed example11.F and example12.c.

4.2 Access for FORTRAN Programmers

FORTRAN programmers have the option of direct navigation of the banks in ZEBRA common, or of using access routines. If using the former method, note must be taken that the contents of the banks may change with the version of the DST, necessitating change to the user code also. For this reason, it is **highly recommended** that consideration be given to using access routines when dealing with the ZEBRA bank contents. If the direct navigation method is chosen, then it is recommended that the parameters which may be obtained by including the file **dstparams.inc** in the source program are used. This is now described.

4.2.1 The include file dstparams.inc

In order to aid in the writing of robust code to fill and use the DST, an include file is provided which contains position parameters of all data words in all banks. By inclusion of this include file in user code, and by using the parameter name to refer to the offset of a given word in a bank rather than the actual integer itself, the user will gain a considerable degree of protection against any changes to the bank structure of the DST which do not involve the removal of words or shuffling between banks. As this include file is generated automatically from the bankdoc for a given DST version using an awk script, the only onus on the user is to ensure the use of the correct version of the file for the version of the DST being read.

Here is a fragment of the **dstparams.inc** file

```
      INTEGER    IRvxsNVert          ,
     +           IRvxsId             ,
     +           IRvxsType           ,
     +           IRvxsNChgd          ,
     +           IRvxsNNeut          ,
```

```
     +           IRvxsNUnused        ,
     +           IRvxsX
      PARAMETER (IRvxsNVert          =  1,
     +           IRvxsId             =  1,
     +           IRvxsType           =  2,
     +           IRvxsNChgd          =  3,
     +           IRvxsNNeut          =  4,
     +           IRvxsNUnused        =  5,
     +           IRvxsX              =  6
```

Instead of obtaining the number of reconstructed vertices with the code

```
      NVERT = IQ(LRVXS+1)
```

one would use

```
      NVERT = IQ(LRVXS+IRvxsNVert)
```

which will remain correct even if the position of the word changes for any reason.

To access the x positions of the vertices, another feature of the include file may be used. Consider the following extract from **dstparams.inc**

```
      INTEGER    NIWHRVXS,   NFWHRVXS,   NIWPRVXS,   NFWPRVXS
      PARAMETER (NIWHRVXS= 1,NFWHRVXS= 0,NIWPRVXS= 5,NFWPRVXS= 4)
      INTEGER    NWHRVXS ,   NWPRVXS
      PARAMETER (NWHRVXS =   NIWHRVXS + NFWHRVXS)
      PARAMETER (NWPRVXS =   NIWPRVXS + NFWPRVXS)
```

The parameter **NWHRVXS** can be seen to contain the number of header words in the RVXS bank, and the parameter **NWPRVXS** to contain the number of data words per vertex. These can be employed when writing the code in the following way

```
      NVERT = IQ(LRVXS+IRvxsNVert)
      IF (NVERT.GT.0) THEN
         DO I = 1,NVERT
            IOFFSET = NWHRVXS + (I-1)*NWPRVXS
            X = Q(LRVXS+IOFFSET+IRvxsX)
         ENDDO
      ENDIF
```

This is more robust than the alternative

```
      NVERT = IQ(LRVXS+1)
      IF (NVERT.GT.0) THEN
         DO I = 1,NVERT
            IOFFSET = 1 + (I-1)*9
            X = Q(LRVXS+IOFFSET+5)
         ENDDO
      ENDIF
```

and would not have to be changed if the number of header words or the position of the word containing x were to change. It is hoped that the advantages of using the parameters in **dstparams.inc** are clear. The parameter names are verbose but quite logical, and also add to the readability of the code.

The file **dstparams.inc** may be found in /nomad/src/dst/v7r4/include, should the user wish to browse it for further possibilities.

4.2.2 Direct Navigation with FORTRAN

Some examples of direct navigation will be given, to illustrate how it can be done.

Example 3 To loop through the RVXS bank, printing the z position of the reconstructed vertices (as was illustrated with C above), could be done with the following code fragment (refer to the bank doc for the bank structure). Note the use of the parameters **IRvxsNVert** and **IRvxsZ** from the include file **dstparams.inc** just described.

```
      SUBROUTINE EXAMPLE3
*.
*........ Code used in example 3 of the documentation
*.
      IMPLICIT NONE
*.
#include "nomzeb.inc"
#include "nkunit.inc"
#include "dstcom.inc"

      INTEGER LDST, LRVXS, NVERT, IOFFSET, I
      REAL    Z
*.
      WRITE(KUNIT6,1000)
*.
*........ Access the DST bank if it exists
*.
      LDST = LQ(LNOMAD-6)
      IF(LDST.GT.0) THEN
*.
*........... Access the RVXS bank if it exists
*.
         LRVXS = LQ(LDST-2)
         IF(LRVXS.GT.0) THEN
*.
*.............. Get the number of vertices in list
*.
            NVERT = IQ(LRVXS+IRvxsNVert)
            IF(NVERT.GT.0) THEN
*.
*................. Loop through the vertices, printing z
*.
```

```
               IOFFSET = NWHRVXS
               DO I = 1,NVERT
                  Z = Q(LRVXS+IOFFSET+IRvxsZ)
                  WRITE(6,1001) Z
                  IOFFSET = IOFFSET + NWPRVXS
               ENDDO
            ENDIF
         ENDIF
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE3 ...')
 1001 FORMAT(' z position of vertex is ',F8.3)
*.
      END
```

Example 4 To loop through the MAS banks, finding only those containing information on muons matched to drift chamber tracks, is more complicated (this example was also given for C above). Here we check if there is information for a muon matched to a drift chamber track, and if so, print its x and y position at station 1.

```
      SUBROUTINE EXAMPLE4
*.
*........ Code used in example 4 of the documentation
*.
      IMPLICIT NONE
*.
#include "nomzeb.inc"
#include "nkunit.inc"
#include "dstparams.inc"

      INTEGER LDST, LMAS, IMUO, IDBLOCK
      REAL    XSTAT1, YSTAT1
*.
      WRITE(KUNIT6,1000)
*.
*........ Access the DST bank if it exists
*.
      LDST = LQ(LNOMAD-6)
      IF(LDST.GT.0) THEN
*.
*........... Loop through the MAS banks
*.
         LMAS = LQ(LDST-3)
         DO WHILE (LMAS.GT.0)
```

```
*.
*............ Check the pointer to muon blocklet
*............ Non-zero means there is muon info
*.
            IMUO = IQ(LMAS+5)
            IF(IMUO.GT.0) THEN
*.
*................ Check the Identifier of this blocklet
*.
               IDBLOCK = IQ(LMAS+IMUO)
               IF(IDBLOCK.EQ.501) THEN
*.
*.................. Found muo-dch blocklet, print x and y at station 1
*.
                  XSTAT1 = Q(LMAS+IMUO+IMasMuoDchXS1)
                  YSTAT1 = Q(LMAS+IMUO+IMasMuoDchXS2)
                  WRITE(6,1001) XSTAT1, YSTAT1
*.
               ENDIF
*.
            ENDIF
*.
            LMAS = LQ(LMAS)
*.
         ENDDO
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE4 ...')
 1001 FORMAT(' x,y position of muon station 1',F8.3,1X,F8.3)
*.
      END
```

4.2.3 Access Routines and FORTRAN

The examples given above can also be tackled with access functions. The C access routines which have been described in section 4.1 have been rendered FORTRAN callable using the cfortran package. Since the C access routines use pointers to the C structures to navigate, one should include the file **pointer.inc** and declare the relevant variables as type **POINTER** in the FORTRAN code.

Example 5 First let us recast the loop through the reconstructed vertices using this method.

```
      SUBROUTINE EXAMPLE5
*.
*....... Code used in example 5 of the documentation
*.
```

```
      IMPLICIT NONE
*.
#include "pointer.inc"
#include "nkunit.inc"
*.
      POINTER DST,  fDstReference,
     +        RVXS, fDstRvxsReference
      INTEGER NVERT, fRvxsNvert, I
      REAL    Z, fRvxsZ
*.
      WRITE(KUNIT6,1000)
*.
*....... Access the DST bank if it exists
*.
      DST = fDstReference()
      IF (DST.NE.0) THEN
*.
*.......... Access the RVXS bank if it exists
*.
         RVXS = fDstRvxsReference(DST)
         IF(RVXS.GT.0) THEN
*.
*............. Get the number of vertices in list
*.
            NVERT = fRvxsNVert(RVXS)
            IF(NVERT.GT.0) THEN
*.
*................ Loop through the vertices, printing z
*.
               DO I = 1,NVERT
                  Z = fRvxsZ(RVXS, I)
                  WRITE(6,1001) Z
               ENDDO
            ENDIF
         ENDIF
      ENDIF
*.
 1000 FORMAT(' Calling EXAMPLE5 ...')
 1001 FORMAT(' z position of vertex is ',F8.3)
*.
      RETURN
      END
```

Example 6 Here is an example of code to access the muon-drift chamber match information using this method.

```
      SUBROUTINE EXAMPLE6
*.
*........ Code used in example 6 of the documentation
*.
      IMPLICIT NONE
*.
#include "pointer.inc"
#include "nkunit.inc"
*.
      POINTER DST,  fDstReference,
     +        MAS,  fDstFirstMasReference, fMasNextReference
      INTEGER fMasBlocklet
      REAL    fMasMuoDchXS1, fMasMuoDchYS1
*.
      WRITE(KUNIT6,1000)
*.
      DST = fDstReference()
      IF (DST.NE.0) THEN
*.
         MAS = fDstFirstMasReference(DST)
         DO WHILE (MAS.GT.0)
*.
            IF(fMasBlocklet(MAS, 501).GT.0) THEN
               WRITE(6,1001) fMasMuoDchXS1(MAS), fMasMuoDchYS1(MAS)
            ENDIF
*.
            MAS = fMasNextReference(MAS)
*.
         ENDDO
*.
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE6 ...')
 1001 FORMAT(' x,y position of muon station 1 is',F8.3,1X,F8.3)
*.
      END
```

4.2.4 Higher-level Access to Bank Contents in FORTRAN

For the same reasons as were discussed in the context of C access functions, higher level access routines are provided to return blocks of information. Such access is provided through the "dstaccess" package [2], which is available in the present DST library.

An alternative set of high level FORTRAN access routines with similar functionality is described below. The existence of the two schemes is historical. They both use the same common block for storage of

variables, "dstaccess.inc". The FORTRAN user should probably choose to settle on one of these schemes and stick to it.

In both schemes, the information requested is placed into a common block which enables the user to access the individual words by name

The names of words in the commons are constructed strictly from the name of the bank or blocklet (in the MAS bank case) and the name of the variable given in the bank documentation. For example the x position of a vertex in the RVXS bank can be referenced through the (real) variable RVXS˙X. The number of hits on a drift chamber track (blocklet MASDCHDCH in the MAS bank) can be referenced through the (integer) variable MASDCHDCH˙NHits. Note that since these are FORTRAN commons, the case of the letters in the name is irrelevant (RVXS˙X and rvxs˙x should work).

In addition to the FORTRAN access routines provided by the "dstaccess" package [2], three additional routines are provided in the library, constituting the second scheme mentioned above. Two, **DSTGetMasBlock** and **DSTGetMasSubBlock** are for use with the MAS bank information, while the third, **DSTGetBlock** is for use with any of the other banks. These routines take three arguments, as follows:

- **Argument 1:** an integer specifying whether the bank address will be passed as a ZEBRA link (1), or as a C pointer (2).
- **Argument 2:** the value of the link or pointer
- **Argument 3:** For the MAS bank, the blocklet ID of the match pair required (e.g. 501 for muon chambers matched to drift chambers). For all other banks, the index of the blocklet required. Note that if the bank has header info, setting this argument to zero causes the header information to be returned.

To use these routines, the user should include in the calling code the include file **dstcom.inc**, and in addition the include file **dstaccess.inc**. **dstcom.inc** provides integer parameters which map to the match pair blocklet IDs, so that one can refer to **MASMUODCH** rather than 501 when making the call to **DSTGetMasBlock**. **dstaccess.inc** contains the common block into which the individual DST words are returned.

Examples will make all of this much clearer.

Example 7 Suppose that one wishes to loop through all of the vertices in the RVXS bank, and that one works with the ZEBRA links to the bank. The following code could be used:

```
      SUBROUTINE EXAMPLE7
*.
*........ Code used in example 7 of the documentation
*.
      IMPLICIT NONE
*.
#include "nkunit.inc"
#include "dstcom.inc"
#include "dstaccess.inc"

      INTEGER   LRVXS, RVXSREF, I
      REAL      X, Y, Z
*.
```

```
      WRITE(KUNIT6,1000)
*.
*....... Get the link to the RVXS bank
*.
      LRVXS = RVXSREF()
      IF (LRVXS.GT.0) THEN
*.
*.......... Unpack the bank header to get the number of vertices
*.
         CALL DSTGetBlock(1, LRVXS, 0)
*.
*.......... Now loop through the vertices if there are any
*.
         IF (Rvxs_NVert.GT.0) THEN
*.
            DO I = 1,Rvxs_NVert
*.
               CALL DSTGetBlock(1, LRVXS, I)
               X = Rvxs_X
               Y = Rvxs_Y
               Z = Rvxs_Z
               WRITE(KUNIT6,1001) I, X, Y, Z
*.
            ENDDO
*.
         ENDIF
*.
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE7 ...')
 1001 FORMAT(' Vertex ',I3,' X = ',F8.3,' Y = ',F8.3,' Z = ',F8.3)
*.
      END
```

Example 8 If instead of using ZEBRA links in the last example, one uses pointers, the code becomes the following:

```
      SUBROUTINE EXAMPLE8
*.
*....... Code used in example 8 of the documentation
*.
      IMPLICIT NONE
*.
#include "nkunit.inc"
```

```
#include "dstcom.inc"
#include "dstaccess.inc"

      POINTER DST, fDstReference, RVXS, fDstRvxsReference
      INTEGER I
      REAL    X, Y, Z
*.
      WRITE(KUNIT6,1000)
*.
*....... Access the DST bank if it exists
*.
      DST = fDstReference()
      IF (DST.GT.0) THEN
*.
*.......... Access the RVXS bank if it exists
*.
         RVXS = fDstRvxsReference(DST)
         IF (RVXS.GT.0) THEN
*.
*............. Unpack the bank header to get the number of vertices
*.
            CALL DSTGetBlock(2, RVXS, 0)
*.
*............. Now loop through the vertices if there are any
*.
            IF (Rvxs_NVert.GT.0) THEN
*.
               DO I = 1,Rvxs_NVert
*.
                  CALL DSTGetBlock(2, RVXS, I)
                  X = Rvxs_X
                  Y = Rvxs_Y
                  Z = Rvxs_Z
                  WRITE(KUNIT6,1001) I, X, Y, Z
*.
               ENDDO
*.
            ENDIF
*.
         ENDIF
*.
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE8 ...')
 1001 FORMAT(' Vertex ',I3,' X = ',F8.3,' Y = ',F8.3,' Z = ',F8.3)
*.
      END
```

Now for two examples using the MAS banks.

Example 9 This code performs the same function as does that of example 4, obtaining information on muon track matched to drift chamber track.

```
      SUBROUTINE EXAMPLE9
*.
*........ Code used in example 9 of the documentation
*.
      IMPLICIT NONE
*.
#include "nomzeb.inc"
#include "nkunit.inc"
#include "dstcom.inc"
#include "dstaccess.inc"

      INTEGER LDST, LMAS, IMUO, IDBLOCK
      REAL    XSTAT1, YSTAT1
*.
      WRITE(KUNIT6,1000)
*.
*........ Access the DST bank if it exists
*.
      LDST = LQ(LNOMAD-6)
      IF(LDST.GT.0) THEN
*.
*.......... Loop through the MAS banks
*.
         LMAS = LQ(LDST-3)
         DO WHILE (LMAS.GT.0)
*.
*............ Check the pointer to the muon blocklet
*............ Non-zero means that it exists
*.
            IMUO = IQ(LMAS+5)
            IF (IMUO.GT.0) THEN
*.
*................ Check the Identifier of this blocklet
*................ Note that parameter MASMUODCH (in dstcom.inc) is 501
*.
               IDBLOCK = IQ(LMAS+IMUO)
               IF(IDBLOCK.EQ.MASMUODCH) THEN
*.
```

```
*.................. Found muo-dch blocklet, print x and y at station 1
*.
                  CALL DSTGetMasBlock(1, LMAS, MASMUODCH)
                  XSTAT1 = MasMuoDch_XS1
                  YSTAT1 = MasMuoDch_YS1
                  WRITE(6,1001) XSTAT1, YSTAT1
*.
               ENDIF
*.
            ENDIF
*.
            LMAS = LQ(LMAS)
*.
         ENDDO
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE9 ...')
 1001 FORMAT(' x,y position of muon station 1',F8.3,1X,F8.3)
*.
      END
```

Example 10 Finally, the previous example using pointers.

```
      SUBROUTINE EXAMPLE10
*.
*........ Code used in example 10 of the documentation
*.
      IMPLICIT NONE
*.
#include "pointer.inc"
#include "nkunit.inc"
#include "dstcom.inc"
#include "dstaccess.inc"
*.
      POINTER DST,  fDstReference,
     +        MAS,  fDstFirstMasReference, fMasNextReference
      INTEGER fMasBlocklet
*.
      WRITE(KUNIT6,1000)
*.
      DST = fDstReference()
      IF (DST.NE.0) THEN
*.
         MAS = fDstFirstMasReference(DST)
         DO WHILE (MAS.GT.0)
*.
            IF(fMasBlocklet(MAS, MASMUODCH).GT.0) THEN
               CALL DSTGetMasBlock(2, MAS, MASMUODCH)
               WRITE(6,1001) MasMuoDch_XS1, MasMuoDch_YS1
            ENDIF
*.
            MAS = fMasNextReference(MAS)
*.
         ENDDO
*.
      ENDIF
*.
      RETURN
*.
 1000 FORMAT(' Calling EXAMPLE10 ...')
 1001 FORMAT(' x,y position of muon station 1 is',F8.3,1X,F8.3)
*.
      END
```

4.3 Utility Routines

Some routines to make use of the DST structure easier are provided, and will be described here. The list will grow with time.

The following C routines are also FORTRAN callable by prepending an f to the routine name, and including **pointer.inc** in the calling FORTRAN code. C programmers should remember to include the file **dstgen.h** in their source code.

4.3.1 General utility routines

- **int *MasBlocklet(˙Mas mas, int blockid)** Returns a non-zero value if the object (MAS bank) pointed to by **mas** contains a blocklet with identifier **blockid** (in fact returns a pointer to the blocklet, of type **˙MasXxxYyy** where **Xxx** is the subdetector supplying the information, e.g. **Trd**, and **Yyy** is the seed subdetector). Otherwise returns 0. This provides a convenient way to check whether the MAS bank contains information from the subdetector of interest. TECHNICAL NOTE: the type of the pointers which are used internally by the C structures which map to the MAS bank blocklets changed from DST version v7r1 to v7r2. In version v7r1, the pointers were of the type **˙MasXxx**, i.e. the pointer was independent of the seed subdetector for the blocklet. This was possible because unions were used in defining the C structures. In versions v7r2 onwards, use of unions had to be abandoned for a technical reason when blocklets could become variable in length, and as a consequence there are now separate pointers to each different type of blocklet, of the form **˙MasXxxYyy**, and the C structures have changed in internal form. Those users who in their C code directly accessed the members of these structures rather than use the access routines provided by the package will find that they have to change their code. This change was unfortunately unavoidable.

- **int NumReconTracks(void)** Returns the total number of reconstructed tracks in the event, obtained by counting the Mas banks containing 101 blocklets.
- **int PrimaryTrack(˙Mas mas)** Returns 1 if the object pointed to by **mas** is attached to the primary vertex (can be charged or neutral), and 0 otherwise.
- **int PrimaryChargedTrack(˙Mas mas)** Returns 1 if the object pointed to by **mas** is charged and attached to the primary vertex, and 0 otherwise.
- **int PrimaryNeutralTrack(˙Mas mas)** Returns 1 if the object pointed to by **mas** is neutral and attached to the primary vertex, and 0 otherwise.

Functions are provided to return certain quantities from the FZ header of the event. The FZ header on a DST tape is presently identical to that on the full recon output tape, so in principle this information is already readily available. For the FORTRAN programmer, the common /NDEVENT/ provided by util may be accessed. For the C programmer, inclusion of the util header file **rzio.h** will allow the structure pointed to by **EVT˙header** to be used. For convenience, the DST library contains the following C access routines for accessing some of the header words. All may be called from FORTRAN by prepending the function name with “f” and declaring the resulting function as INTEGER.

- **int DstRunNumber()** Returns the current run number.
- **int DstEventNumber()** Returns the current event number.
- **int DstTrigMask()** Returns the undelayed trigger mask (as a bitted word, in which bits 11 to 16 contain the relevant information). This function simply uses the util function RzGetTriggerMask().
- **int DstTrigMaskDel()** Returns the delayed trigger mask (as a bitted word, in which bits 27 to 32 contain the relevant information). This function simply uses the util function RzGetTriggerMaskInt().
- **int DstOnlineError()** Returns the online error word.
- **int DstOfflineError()** Returns the offline error word.

The following two routines are provided to aid calculation of the magnitude and error of the 3-momentum of an object given the components. Whilst the first is trivial, the second may be of use when the upper diagonal of the covariance matrix for (p_x, p_y, p_z) is available. It uses the routine **track˙cartesian** from the extrapolator package.

- **DstPxPyPzToP(float *p)** Returns the magnitude of the 3-momentum given a pointer to the word containing p_x (and assuming the 3 components are stored contiguously).
- **DstCovPxPyPzToEP(float *p, float *ep)** Returns the error on the magnitude of the 3-momentum given pointers to the word containing p_x and the word containing $Cov(p_x, p_x)$ (and assuming the 3 momentum components, and the 6 upper diagonal elements of the covariance matrix, are stored contiguously, as they are on the DST).

4.3.2 Extended access routines

Also available are some routines to return useful quantities such as momentum and error on momentum *as though* these quantities were stored in the DST structure (they have in fact been removed from the structure from DST v7r1 onwards as they are calculable from the quantities which are stored there). The list is (all are of type **float**):

- **MasDchDchPB(˙Mas Mas)** Momentum of track at beginning vertex
- **MasDchDchEPB(˙Mas Mas)** Error on momentum of track at beginning vertex
- **MasDchDchPF(˙Mas Mas)** Momentum of track at first hit
- **MasDchDchEPF(˙Mas Mas)** Error on momentum of track at first hit
- **MasDchDchPL(˙Mas Mas)** Momentum of track at last hit
- **MasDchDchEPL(˙Mas Mas)** Error on momentum of track at last hit
- **MasDchDchPE(˙Mas Mas)** Momentum of track at end vertex
- **MasDchDchEPE(˙Mas Mas)** Error on momentum of track at end vertex
- **StksPB (˙Stks Stks, int i)** Momentum of simulated track **i** at beginning vertex
- **StksPF (˙Stks Stks, int i)** Momentum of simulated track **i** at first hit
- **StksPL (˙Stks Stks, int i)** Momentum of simulated track **i** at last hit
- **StksPE (˙Stks Stks, int i)** Momentum of simulated track **i** at end vertex

4.3.3 Access routines for packed words

As some of the words in the DST structure are packed using base 10 multiplicative factors, routines are provided which look like the standard access routines to the packed word as a whole, but which extract particular quantities from that word. By inspecting the bankdoc (or chapter 5) for a description of the contents of the relevant words, the function of the following verbosely named access routines should become self-evident.

For unused hits around a vertex in the RVXS bank:

- **int RvxsNUnusedU15 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedU10 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedU05 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedY15 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedY10 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedY05 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedV15 (˙Rvxs Rvxs, int i)**

- **int RvxsNUnusedV10 (˙Rvxs Rvxs, int i)**
- **int RvxsNUnusedV05 (˙Rvxs Rvxs, int i)**

For packed muon chamber hits in the 101 blocklet:

- **int MasDchDchMuHitsX1 (˙Mas Mas)**
- **int MasDchDchMuHitsX2 (˙Mas Mas)**
- **int MasDchDchMuHitsY1 (˙Mas Mas)**
- **int MasDchDchMuHitsY2 (˙Mas Mas)**
- **int MasDchDchMuHitsMuV (˙Mas Mas)**
- **int MasDchDchMuHitsNFX1 (˙Mas Mas)**
- **int MasDchDchMuHitsNFX2 (˙Mas Mas)**
- **int MasDchDchMuHitsNFY1 (˙Mas Mas)**
- **int MasDchDchMuHitsNFY2 (˙Mas Mas)**

For packed drift chamber veto and forward and backward hits for a track in the 101 blocklet:

- **int MasDchDchNDcVHitBTube (˙Mas Mas)**
- **int MasDchDchNDcVHitBU50 (˙Mas Mas)**
- **int MasDchDchNDcVHitBY50 (˙Mas Mas)**
- **int MasDchDchNDcVHitBV50 (˙Mas Mas)**
- **int MasDchDchNDcVHitBU25 (˙Mas Mas)**
- **int MasDchDchNDcVHitBY25 (˙Mas Mas)**
- **int MasDchDchNDcVHitBV25 (˙Mas Mas)**
- **int MasDchDchNDcVHitFTube (˙Mas Mas)**
- **int MasDchDchNDcVHitFU50 (˙Mas Mas)**
- **int MasDchDchNDcVHitFY50 (˙Mas Mas)**
- **int MasDchDchNDcVHitFV50 (˙Mas Mas)**
- **int MasDchDchNDcVHitFU25 (˙Mas Mas)**
- **int MasDchDchNDcVHitFY25 (˙Mas Mas)**
- **int MasDchDchNDcVHitFV25 (˙Mas Mas)**

4.3.4 Routines for backwards compatibility

The following are provided for backward compatibilty. The word names were changed in MAS blocklet 501 in going from versions v1r0 to v7r1 of the library, in order to be more consistent with other blocklets (basically the letters **Mu** were dropped, e.g. **MasMuoDchXMuS1** became **MasMuoDchXS1**). The functions below, which utilise the old form of the word, will if used return the new word.

- **float MasMuoDchXMuS1 (˙Mas Mas)**
- **float MasMuoDchYMuS1 (˙Mas Mas)**
- **float MasMuoDchZMuS1 (˙Mas Mas)**
- **float MasMuoDchXMuS2 (˙Mas Mas)**
- **float MasMuoDchYMuS2 (˙Mas Mas)**
- **float MasMuoDchZMuS2 (˙Mas Mas)**
- **float MasMuoDchSlXMuS1 (˙Mas Mas)**
- **float MasMuoDchSlYMuS1 (˙Mas Mas)**
- **float MasMuoDchSlXMuS2 (˙Mas Mas)**
- **float MasMuoDchSlYMuS2 (˙Mas Mas)**

Chapter 5: The Banks in More Detail

The general policy in filling the DST is to take as much information as possible exactly "as is" from the phase2 ZEBRA structure. The contents of the vast majority of the words can be deduced from the information in the bankdoc. However, in some cases further explanation and clarification of what is in a word is required. In this chapter all words on the DST are described, in more detail than the bankdoc where necessary. For each word, the description includes an indication of whether the word is integer (I) or floating point (F).

5.1 The FZ Header

The FZ Header that is present for all ZEBRA files throughout production is preserved and propagated to DST ZEBRA files as well. Therefore these words are available to users along with the contents of the DST structure itself – indeed this is the way that most people obtain the run and event number, which are not stored in the DST banks themselves.

Access to the information in the FZ header may be obtained in one of several ways.

- For a FORTRAN programmer, the include file **ndevent.inc** may be used. This contains the common block /NDEVENT/, into which many of the words in the FZ header have been copied upon standard input of an event with a program such as **camel**. For example, the run and event numbers are available in the words **IRUN** and **IEVENT**.
- Also for a FORTRAN programmer, the include file **nzlink.inc** may be used (this is also obtained if the file **nzbank.inc** has been included). The word **LHEAD** holds the ZEBRA link to the FZ header, and words are obtained by referencing the IQ array using this link. For example, the run and event numbers via this method may be found in **IQ(LHEAD+1)** and **IQ(LHEAD+2)**. **All** header words can be accessed in this way.
- For C programmers, the header file **rzio.h** (from the util package) may be used. This provides access to a structure pointed to by **EVT˙header**. To get the run and event numbers, the appropriate members of that structure, **EVT˙header->run˙number** and **EVT˙header->event˙number** can be accessed.
- A very limited subset of these words can be accessed using DST library access functions. See chapter 4 for more details.

There are two important event quality words, present in the FZ header of an event, which users should be aware of. These are

- **IQ(LHEAD+24)** The so-called "online error" word. This will be non-zero if there is raw data missing in the event for some of the subdetectors. C programmers following the prescription above will find this word with **EVT˙header->online˙error**.
- **IQ(LHEAD+29)** The so-called "offline error" word. This will be non-zero if the event has failed the "density cuts" or if there have been problems in the matching process. Since in these cases full reconstruction will not have been done but the event may still appear in the output data set, it is important to check this word. Its actual value conveys information on which condition caused it to fail.

- **Bit 0 Set** Failed the drift chamber veto cut.
- **Bit 1 Set** Failed the cut on the number of drift chamber hits.
- **Bit 2 Set** Failed the wire density cut.
- **Bit 3 Set** Failed the cut on the number of TRD hits.
- **Bit 16 Set** Problem with the matching process (for example, failed to converge).

Note that C programmers can currently find this word with the inappropriately named **EVT˙header->online˙stream** (at least up to util v7r5). This is because the word was originally earmarked for another use, and has subsequently been appropriated. A name change in future util versions is called for.

5.2 The DST bank

The DST bank is simply a hanger onto which the other banks in the DST structure are attached. As such, it contains just one data word:

- **Version** The DST version number, a floating point number containing the version and release number. (F)

The version number may be obtained using

```
(int) Version
```

The release number may be obtained with

```
(int) 10*(Version - (int) Version)
```

Remember when comparing the word itself with a known version number (e.g. 7.4) to be wary of roundoff. It is better to extract the integer version and release as above and look at those.

5.3 The EVS bank

The purpose of the EVS bank is to provide some global information on the event. Some of the information, as pointed out below, is meant to be indicative only, meaning that to make sophisticated cuts, especially on the energy variables, the user is advised to access the detailed information elsewhere on the DST.

- **IndxPrim** All words on the DST which are advertised as "indices" give the index of an object in another bank. If one considers the RVXS bank to consist of a table where each row contains information on one reconstructed vertex, then **IndxPrim** tells the user which row to look at in order to find information on the primary vertex for the event. If this word is zero, there is no primary. (I)
- **NVert** This is simply the number of vertices in the event with more than one track emerging, with the exception that the primary is always counted, even if it has only one track. Vertices may have a charged or neutral track incident. The information is obtained by examining the RVXS bank. (I)
- **NHang** All reconstructed tracks with beginning vertex of type 9 are counted. (I)
- **NChgd** All charged tracks (MAS blocklets 101) from primary are counted. (I)
- **NNeut** All neutral tracks (MAS blocklets 104) from primary are counted. (I)

- **NCalUn** All neutral CAL clusters (MAS blocklets 404) which are not accompanied by a neutral track (MAS blocklet 104) are counted. (I)
- **NHcaUn** All unassociated HCAL clusters (MAS blocklets 909) are counted. (I)
- **NTrig1** Counts the number of in-time hits in plane 1 of the trigger counters. Since there is no match done for the trigger counters, the hits in the DASC bank are used. The time windows are taken from the RHST bank in the run header structure for the trigger. (I)
- **NTrig2** The same as **NTrig1** only for trigger plane 2. (I)
- **NVeto** Counts the number of in-time hits for the veto counters (the "real" veto, not the first drift chamber). Uses the information in MAS blocklets 701 and 707. (I)
- **NCells** Counts the number of calorimeter cells above threshold. Uses the information in the phase1 DACA bank (no additional threshold above that used to fill the DACA bank is imposed). (I)
- **NSlabs** Counts the total number of HCAL slabs which are hit. This is achieved by accessing the information in the DAHC bank and finding those slabs with non-zero energy deposition. (I)
- **NMuMat** This word gives the number of "good" muons in the event. It is filled by counting all charged tracks in the event which have the word **ProbMu** in the 101 blocklet (see below) set to 1.0. Note that this behaviour is *different* to that in version v7r1 of the DST, where the word had the name **NMuTrk** and was filled by counting 501 blocklets rather than "good" muons. This was also the case for version v1r0 DSTs. Note that situations in which 2 muons are flagged in this word should not be treated as "golden" dimuons on the present DST, as the ambiguity treatment at the level of phase2 is not fully developed. (I)
- **NRemain** The number of drift chamber hits which were left over after track reconstruction was completed is stored, the information being taken from word 5 of the DDCH bank. (I)
- **Overflow** This word is a copy of word 12 of the SEVT bank. For Monte Carlo events, it flags the condition that a buffer overflow during the GENOM processing of this event has caused information to be lost. (I)
- **NDcVeto** The number of hits in the drift chamber veto chamber. This and the following three words are used in performing the filtering during processing. They are taken from the corresponding words in the BDCH bank. (I)
- **NDcHit** The total number of (in-time) hits in all drift chambers. (I)
- **NDensity** The wire density, multiplied by 1000. (I)
- **NTrdHit** The total number of TRD hits. (I)
- **ECells** The energy deposited in all the cells stored in the phase1 DACA bank is summed. (F)
- **ESlabs** The energy deposited in all the scintillators stored in the DAHC bank is summed. (F)
- **Weight** The Monte Carlo weight assigned to this event. Since no attempt has yet been made at the production stage to combine event samples from different generators into a single data set, this weight is always 1.0. (F)

5.4 The RVXS bank

The RVXS bank is essentially a summary in tabular form of the information in the VTX structure. All vertices which are present in the VTX structure are present in the RVXS bank with one exception: end point vertices (those with vertex type 10) are **not** stored except when there is a neutral associated to the vertex – this happens in phase2 at a low level and so some type 10 vertices may be found on the DST.

- **NVert** Indicates how many vertices are stored in the bank.

For each vertex, the following words are stored.

- **Id** The vertex identifier stored in word 20 of the VTX bank. (I)
- **Type** The vertex type stored in word 21 of the VTX bank. (I) Valid types are:

  : **1** Primary vertex
  : **2** Secondary
  : **3** Decay (one track in, several out)
  : **4** V0
  : **5** Brem
  : **6** Scatter (one track in, one track out)
  : **7** Delta Ray
  : **8** Hard Scatter
  : **9** Beginning (hanger)
  : **10** End
  : **11** Neutral Hanger (do not appear on DST)
  : **12** Out

- **NChgd** Counts all charged tracks associated to the vertex (by looping over the TRK banks). (I)
- **NNeut** Counts all neutral tracks associated to the vertex (by looping over the TRK banks). (I)
- **NUnused** Unused hits in the vicinity of the vertex. This is a packed word, filled as follows:

```
  1000000*(No. Hits in box of size +- 15cm)
+    1000*(No. Hits in box of size +- 10cm)
+         (No. Hits in box of size +-  5cm)
Each "No. Hits in box" = 100*(No. U Hits) (0-9)
                 +        10*(No. Y Hits) (0-9)
                 +           (No. V Hits) (0-9)
```

  In this context, a "box" of size l extends in z a distance l before and after the z position of the vertex. For the other dimensions, a road of the same size is defined, $\pm l$, oriented separately with respect to each of the 3 orientations of the wires (U,Y,V) for the purpose of counting unused hits within the road. Access routines to unpack the different components of this word are provided - see section 4.3.3. (I)

- **X** The x position of the vertex. (F)
- **Y** The y position of the vertex. (F)
- **Z** The z position of the vertex. (F)
- **Chi2** $\chi 2$ of vertex fit. (F)
- **Chi2MisM** χ^2 for the hypothesis that a V0 "points" to the primary vertex (2 d.o.f.). The vertex-fitting package constructs a "mismatch vector" $\vec{v}$ between a projected V0 track and the primary vertex at the point of closest approach: "pointing" is equivalent to the hypothesis that this vector $\vec{v} = \vec{0}$ (Memo 96-019, section 8). This word has been added to the DST (in v7r4) because the necessary calculations use both track and vertex error matrices, and the latter are not recorded to the DST. This word is identically zero for non-V0 vertices.

5.5 The MAS bank

The MAS banks summarise the information in the match table produced by phase 2 processing, there being essentially one MAS bank for each object, or line of the match table.

The first set of words in the MAS bank are so-called "pointers" to the information stored in the bank for each subdetector and "pseudo" subdetector. A subdetector is a physical NOMAD subdetector, of which there are 10 for the purposes of the present DST (1=DCH, 2=TRD, 3=PRS, 4=CAL, 5=MUO, 6=SCI, 7=VET, 8=FCA, 9=HCA, 10=SIT). The SIT of "NOMAD Star" pointer is always empty in the present DST version. "Pseudo" subdetectors are blocklets with identifiers greater than 10 and are used to store additional information about a match object in a manner that can be handled similarly to the "real" subdetector match information. Pseudo subdetectors in the context of the present DST are 11=Feldman Clustering, 12=Extrapolator information and 13=Padova Bremsstrahlung Strip information.

If a pointer to a given subdetector is non-zero, then that subdetector has contributed information to this object. The value of the pointer is the offset within the bank of the word containing the blocklet type for the subdetector – subsequent words in the bank give the data for that blocklet. There can only be one blocklet per subdetector in any given object. For example, if the pointer to TRD info is non-zero, the blocklet pointed to will be a 201 (TRD track matched to drift chamber track) or a 202 (TRD standalone track) but not both.

The information stored for each blocklet will now be covered in some detail. The order of the words in the bank has not necessarily been preserved in this discussion, since occasionally they are grouped by subject.

5.5.1 Blocklet 101

The 101 blocklets store information about reconstructed charged tracks. The majority of these words are derived directly from the TRK bank; it is indicated where this is not the case.

- **DchId** The track reconstruction number (this corresponds to the number provided by **DcTrackRec** in the drift chamber package). (I)
- **IndxVxsB** This word gives the index in the RVXS bank of the beginning vertex for this track. On the DST, an "index" can be thought of as a row number, or line, in a table. If one considers the RVXS bank to consist of a table where each row contains information on one reconstructed vertex, then **IndxVxsB** tells the user which row to look at in order to find information on the beginning vertex for this track. (I)

- **IndxVxsE** Index of end vertex in RVXS bank. See the previous discussion of **IndxVxsB**. Note that this word may be set to zero in cases where the end point vertex type is 10, since these vertices are not retained in the RVXS bank. (I)
- **IndxStks** Index of simulated track in STKS bank. This can only be non-zero for Monte Carlo data. The TRK banks contain a reference link to the corresponding STRK bank containing the simulated track, where a correspondence has been established. This word is filled by extracting the ID of the simulated track from the appropriate STRK bank, and matching it with the entries in the STKS bank. This word will not be filled in *all* cases. (I)
- **NHits** The number of hits on the track. (I)
- **NDF** The number of degrees of freedom used for the track fit. (I)
- **Charge** The charge of the track. (I)
- **Type** Geant particle code of model used in the the track fit. Note that the fact that this stores the code corresponding to the **model** means that for both electrons and positrons this word will have the code for an electron (since the electron model is used for fitting in both cases). Thus this word should not be used to determine the charge of the particle - **Charge** should be used instead. (I)
- **NDcVHitB** A packed word giving information on hits in the veto drift chamber around the point at which an extrapolation of the track backwards intersects it, for two given radii. Also the number of hits collected in a road backwards from the start of the track is included. The format is as follows:

```
  1000000*(No. Hits in Tube 3cm wide, 50cm long)
+    1000*(No. Hits in Dc Veto, 5.0cm radius circle)
+         (No. Hits in Dc Veto, 2.5cm radius circle)
No. Dc Veto Hits = 100*(No. U Hits)       (0-9)
                 +  10*(No. Y Hits)       (0-9)
                 +     (No. V Hits)       (0-9)
```

  As this information is not stored in the current phase2 output structure, the DST code obtains this information by using code provided by A. Geiser and A. Bueno, which accesses the DADC bank. Access routines to unpack the different components of this and the following word are provided - see section 4.3.3 (I).
- **NDcVHitF** A packed word giving information of the same nature as the previous word **NDcVHitB**, only in this case the track is followed forward from its end point. (I)
- **Chi2** The χ^2 of the track fit. (F)
- **ProbChi2** The χ^2 probability of the track fit. (F)

As a result of the fitting procedure for tracks and vertices, the momentum (and associated error matrix) for a track is determined at several points: at the plane of the first and last drift chamber hits associated with the track, and at the beginning and end point vertices which the track connects. If the track is a hanger or leaves the chamber, the first or last hit may of course correspond to the beginning or last vertex. The TRK banks store, at the beginning and end vertices of the track, the fitted 3-momentum and

the 6 elements of the upper diagonal of the covariance matrix for the 3-momentum. At the first and last hits, the quantities stored are $1/p$, x, y, t_x, t_y and corresponding upper diagonal of the covariance matrix. t_x and t_y are the slopes of the track in the $x-z$ and $y-z$ plane at the point (x, y, z). For consistency, on the DST the first and last hit parameters are transformed to p_x, p_y, p_z, using utility routines from the extrapolator package.

- **PxB** The x component of the momentum at the beginning vertex. (F)
- **PyB** The y component of the momentum at the beginning vertex. (F)
- **PzB** The z component of the momentum at the beginning vertex. (F)
- **EPxPxB** $V_{p_x} = cov(p_x, p_x)$ at the beginning vertex. (F)
- **EPxPyB** $cov(p_x, p_y)$ at the beginning vertex. (F)
- **EPxPzB** $cov(p_x, p_z)$ at the beginning vertex. (F)
- **EPyPyB** $V_{p_y} = cov(p_y, p_y)$ at the beginning vertex. (F)
- **EPyPzB** $cov(p_y, p_z)$ at the beginning vertex. (F)
- **EPzPzB** $V_{p_z} = cov(p_z, p_z)$ at the beginning vertex. (F)
- **PxF** The x component of the momentum at the first hit. (F)
- **PyF** The y component of the momentum at the first hit. (F)
- **PzF** The z component of the momentum at the first hit. (F)
- **EPxPxF** $V_{p_x} = cov(p_x, p_x)$ at the first hit. (F)
- **EPxPyF** $cov(p_x, p_y)$ at the first hit. (F)
- **EPxPzF** $cov(p_x, p_z)$ at the first hit. (F)
- **EPyPyF** $V_{p_y} = cov(p_y, p_y)$ at the first hit. (F)
- **EPyPzF** $cov(p_y, p_z)$ at the first hit. (F)
- **EPzPzF** $V_{p_z} = cov(p_z, p_z)$ at the first hit. (F)
- **PxL** The x component of the momentum at the last hit. (F)
- **PyL** The y component of the momentum at the last hit. (F)
- **PzL** The z component of the momentum at the last hit. (F)
- **EPxPxL** $V_{p_x} = cov(p_x, p_x)$ at the last hit. (F)
- **EPxPyL** $cov(p_x, p_y)$ at the last hit. (F)
- **EPxPzL** $cov(p_x, p_z)$ at the last hit. (F)
- **EPyPyL** $V_{p_y} = cov(p_y, p_y)$ at the last hit. (F)

- **EPyPzL** $cov(p_y, p_z)$ at the last hit. (F)
- **EPzPzL** $V_{p_z} = cov(p_z, p_z)$ at the last hit. (F)
- **PxE** The x component of the momentum at the end vertex. (F)
- **PyE** The y component of the momentum at the end vertex. (F)
- **PzE** The z component of the momentum at the end vertex. (F)
- **EPxPxE** $V_{p_x} = cov(p_x, p_x)$ at the end vertex. (F)
- **EPxPyE** $cov(p_x, p_y)$ at the end vertex. (F)
- **EPxPzE** $cov(p_x, p_z)$ at the end vertex. (F)
- **EPyPyE** $V_{p_y} = cov(p_y, p_y)$ at the end vertex. (F)
- **EPyPzE** $cov(p_y, p_z)$ at the end vertex. (F)
- **EPzPzE** $V_{p_z} = cov(p_z, p_z)$ at the end vertex. (F)

The spatial position of the first and last hits on each track are stored, along with the errors on the position of the first hit. Also, the track length using the first and last hits is available in the TRK bank, and propagated to the DST.

- **XF** The x position of the first hit on the track. (F)
- **YF** The y position of the first hit on the track. (F)
- **ZF** The z position of the first hit on the track. (F)
- **EXXF** $cov(x, x)$ at the first hit on the track. (F)
- **EXYF** $cov(x, y)$ at the first hit on the track. (F)
- **EYYF** $cov(y, y)$ at the first hit on the track. (F)
- **XL** The x position of the last hit on the track. (F)
- **YL** The y position of the last hit on the track. (F)
- **ZL** The z position of the last hit on the track. (F)
- **Length** Track length (cm). (F)

Both the purity and efficiency words which characterize the match between a reconstructed and simulated track in Monte Carlo events are now filled in the TRK banks and can be transferred to the DST.

- **Purity** Purity of match to simulated track. (F)
- **Efficm** Efficiency of match to simulated track. (F)

For those tracks which have been identified as being an electron, a refit of the track using the electron hypothesis is performed by the reconstruction, and the track parameters recorded in the TRK bank will be those of the electron fit. In order to have available some momentum information assuming the track was a pion, in such cases the magnitude of the momentum from the pion fit is stored in the TRK bank and transferred to word **PPion** on the DST. If the electron fit was not employed, the word **PPion** will contain 0.0, as in the TRK bank.

- **PPion** Momentum from pion fit. (F)

The following six words give breakpoint information for the track. For a detailed discussion of their contents, MEMO 96-016 [4] should be consulted. Note that the last four words only contain useful information in those cases where the number of hits on the track, **NHits**, is greater than 21. In other cases, **FChisq7** and **FChisq9** contain 10000000. and **Diff7Rm1** and **Diff9Rm1** contain -9999.0 (as in the TRK bank).

- **Ck** Mismatched χ^2. (F)
- **Fk** Fruhwirth. (F)
- **FChisq7** Fisher F7. (F)
- **Diff7Rm1** Back-front 1/R difference in σ for the 7 parameter case. (F)
- **FChisq9** Fisher F9. (F)
- **Diff9Rm1** Back-front 1/R difference in σ for the 9 parameter case. (F)

The final word in the 101 blocklet gives the t_0 of the track at the first hit. It is stored primarily to aid in studies of the FCAL, where timing is not as accurately determined and tracks in the drift chambers emerging from the FCAL will not have a well determined vertex.

- **TZero** The t_0 of the track at the first hit. (F)

The muon information which is stored in the 101 blocklet is derived directly from the DMMU bank. Apart from the words **ProbMu** and **ProbMuH**, the other words come from the muon veto blocklet (the muon veto blocklet in the DMMU bank has ID of –1).

- **MuHits** This word summarises the hits in the x and y projections in each station for this track, as well as the hits in the muon veto scintillators. It is constructed as follows: if the number of hits in projection and station are denoted **NX1**, **NY1**, **NX2**, **NY2** (with obvious notation), and **NV** is the sum of the hits in each of the 4 muon veto scintillators, then $\mathbf{MuHits} = \mathbf{10000} \times \mathbf{NX2} + \mathbf{1000} \times \mathbf{NY2} + \mathbf{100} \times \mathbf{NX1} + \mathbf{10} \times \mathbf{NY1} + \mathbf{NV}$. The relevant words in the muon veto blocklet are 8 and 9. Note that unfortunately the ordering of station and projection used in constructing this word differs from that used in word 9 of the muon veto blocklet in the DMMU bank. Note also that the muon veto scintillators were not present in 1995. Access routines to unpack the different components of this and the following word are provided - see section 4.3.3 (I).
- **MuHitsNF** This word is in the same format as the previous word, **MuHits**, only in this case only hits which have not been flagged as belonging to a matched muon are counted. Thus the format is $\mathbf{MuHits} = \mathbf{10000} \times \mathbf{NX2} + \mathbf{1000} \times \mathbf{NY2} + \mathbf{100} \times \mathbf{NX1} + \mathbf{10} \times \mathbf{NY1}$ In this case, as the information is not stored in the current phase2 output structure, the DST code calls the muon library routine CountMuNotFl to obtain the information. (I)

- **ProbHit1** Probability to hit muon station 1. (F)
- **ProbHit2** Probability to hit muon station 2. (F)
- **ProbHitG** Probability to hit gap in muon chambers in station 1. (F)
- **ProbHitV** Probability to hit muon veto counters. (F)
- **ProbRch1** Probability to reach muon station 1. (F)
- **ProbRch2** Probability to reach muon station 2. (F)
- **ProbMu** Probability to be a muon from muon chambers. This word is set from the global quality word (word 5) of the DMMU bank. If the global quality word is 0 (identified muon) then **ProbMu** is set to 1.0. The information contained in this word is presently equivalent to that in the word **Phmu** in the DMMU bank muon veto blocklet, and also flagging of a muon with this word should correspond to the GEANT particle code for the track being set to 5 or 6. (F)

The word **ProbMuH** is the result of the output of the algorithm due to Gary Feldman which sets out to identify muons using information from ECAL and HCAL. This is a direct copy of the corresponding word in the phase2 DMHC bank.

- **ProbMuH** Probability to be a muon from ECAL/HCAL information. (F) The fractional part of this word gives the combined probability using ECAL and HCAL. It is set to 0.5 if information from neither is available. The integer part of the word is a status code, having the following possible values:
  - : **0** Both ECAL and HCAL information used.
  - : **1** Only HCAL information used. ECAL had zero energy or potential overlaps.
  - : **2** Only ECAL information used. HCAL had low fiducial coverage or potential overlaps.
  - : **3** Neither ECAL and HCAL information used.

5.5.2 Blocklet 104

The 104 blocklets store information about neutral "tracks", i.e. calorimeter objects which have been associated to a reconstructed vertex. In the phase2 output structure, the information for these tracks are stored in TRK banks hanging from dummy vertices of type 11.

In the version of phase2 output from which this DST is derived, association of calorimeter objects is only made to the primary vertex (although in a few cases one will find objects associated to end point vertices of type 10). This situation should be contrasted with that for DST v7r1, where associations to many different vertex types appeared.

- **DchId** The track reconstruction number (this is analogous to the number provided by **DcTrackRec** in the drift chamber package for charged tracks). (I)
- **IndxVxsB** This word gives the index in the RVXS bank of the beginning vertex for this track. On the DST, an "index" can be thought of as a row number, or line, in a table. If one considers the RVXS bank to consist of a table where each row contains information on one reconstructed vertex, then **IndxVxsB** tells the user which row to look at in order to find information on the beginning vertex for this track. (I)

- **IndxVxsE** Index of end vertex in RVXS bank. See the previous discussion of **IndxVxsB**. For neutral tracks as implemented in the present DSTs, this word will always be set to zero, since all neutral tracks are constructed from calorimeter objects and so do not have an end point vertex in the RVXS bank. (I)
- **IndxStks** Index of simulated track in STKS bank. This can only be non-zero for Monte Carlo data. The TRK banks contain a reference link to the corresponding STRK bank containing the simulated track, where a correspondence has been established. This word is filled by extracting the ID of the simulated track from the appropriate STRK bank, and matching it with the entries in the STKS bank. This word will not be filled in *all* cases. (I)
- **Type** Geant particle code for this track. This will be type 1, corresponding to a photon. (I)
- **PxB** The x component of the momentum at the beginning vertex. (F)
- **PyB** The y component of the momentum at the beginning vertex. (F)
- **PzB** The z component of the momentum at the beginning vertex. (F)
- **EPxPxB** $V_{p_x} = cov(p_x, p_x)$ at the beginning vertex. (F)
- **EPxPyB** $cov(p_x, p_y)$ at the beginning vertex. (F)
- **EPxPzB** $cov(p_x, p_z)$ at the beginning vertex. (F)
- **EPyPyB** $V_{p_y} = cov(p_y, p_y)$ at the beginning vertex. (F)
- **EPyPzB** $cov(p_y, p_z)$ at the beginning vertex. (F)
- **EPzPzB** $V_{p_z} = cov(p_z, p_z)$ at the beginning vertex. (F)

The track length for a neutral track is not stored in the phase2 output, but is calculated for the DST as follows. The position in space of the shower corresponding to the calorimeter object that this track represents is extracted from the DMCA bank, and the modulus of the vector joining this space point to the primary vertex is taken as the length of the track.

- **Length** Track length (cm). (F)

Note that the spatial position that is stored in the 404 blocklet that accompanies a 104 blocklet is the position of the ECAL *cluster* which has been used in forming this track, not the position of the *shower*. The latter is not stored on the present DSTs, and if required, must be calculated from the position of the primary vertex, the direction of the track as given by the unit vector derived from the 3-momentum of the track, and the track length.

5.5.3 Blocklet 201

The 201 blocklets store information for TRD tracks which have been matched to a drift chamber track. The information is taken directly from the DMTR and DOTR banks, with the exception of the average energy deposition per plane, for which case the energy associated with each hit is taken from the DATR bank.

For more information on the words in this blocklet, the NOMAD memos 95-041 [6] and 96-005 [7] could be consulted. Some of the description below is derived from the phase 2 Web pages [5].

- **TrdId** The TRD track identifier. (I)
- **NHits** The number of TRD planes hit for this track. (I)
- **DoubP** The overlap flag. (I) The values taken are as follows:
  - : **-1** Isolated track.
  - : **-11** Single Id applied because double was not applicable.
  - : **0** Pion after 1 track and and 2 track Id.
  - : **1** Electron after 1 track changed to pion after 2 track Id.
  - : **10** Pion after 1 track changed to electron after 2 track Id.
  - : **11** Electron after 1 track and and 2 track Id.
- **NOver** The number of overlapping tracks. (I)
- **NShit** The number of shared hits for this track. (I)
- **NIden** The indecision flag. This is set to 1 if the result of 2d track identification was electron plus pion and both have $p > 2\ \mathrm{GeV}/c$. In this case it is hard to take a decision. (I)
- **Dist** Mismatch parameter between TRD and DCH tracks. (F) This is taken from the DMTR bank, where it is filled with the value returned by the function **DcTrdDistance** (this is in **trdmatch.c** in the **match** subdirectory of the phase2 source).
- **EAvg** The average energy deposition per plane (in keV). The energy per hit is obtained from the DATR bank. In cases where there has been an ADC overflow, the energy word in the DATR bank has been set artificially to 99999 keV, and for the present calculation, a value of 100 keV is taken. (F)
- **ProbEl** Probability to be an electron. (F)
- **PionCon** The pion contamination. If the algorithm was not applied, the value -1 will be found. (F)
- **ElAcc** The electron acceptance. (F)
- **PionCtr** The pion contamination (truncated). (F)
- **ElAtr** The electron acceptance (truncated). (F)
- **ProtCon** The proton contamination. (F)
- **EHit1** The energy deposition in plane 1 (keV; $\mathbf{EHit1} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit2** The energy deposition in plane 2 (keV; $\mathbf{EHit2} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit3** The energy deposition in plane 3 (keV; $\mathbf{EHit3} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit4** The energy deposition in plane 4 (keV; $\mathbf{EHit4} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit5** The energy deposition in plane 5 (keV; $\mathbf{EHit5} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit6** The energy deposition in plane 6 (keV; $\mathbf{EHit6} < \mathbf{0}$ denotes a shared hit). (F)

- **EHit7** The energy deposition in plane 7 (keV; $\mathbf{EHit7} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit8** The energy deposition in plane 8 (keV; $\mathbf{EHit8} < \mathbf{0}$ denotes a shared hit). (F)
- **EHit9** The energy deposition in plane 9 (keV; $\mathbf{EHit9} < \mathbf{0}$ denotes a shared hit). (F)

The words **EHit1** through **EHit9** were introduced in DST v7r3, where they were always positive numbers. In the DST v7r4, however, a *negative* number is used to flag a straw tube “hit” which is shared with one or more other DCH-TRD matched tracks. To obtain the energy deposited, the absolute value should be taken.

5.5.4 Blocklet 202

The 202 blocklets store information for TRD standalone tracks i.e. those tracks not matched to a drift chamber track. The information is taken directly from the DOTR banks, with the exception of the average energy deposition per plane, for which case the energy associated with each hit is taken from the DATR bank.

- **TrdId** The TRD track identifier. (I)
- **NHits** The number of TRD planes hit for this track. (I)
- **EAvg** The average energy deposition per plane (in keV). The energy per hit is obtained from the DATR bank. In cases where there has been an ADC overflow, the energy word in the DATR bank has been set artificially to 99999 keV, and for the present calculation, a value of 100 keV is taken. (F)
- **B** The track intercept parameter B ($\mathrm{x} = \mathrm{A} \times \mathrm{z} + \mathrm{B}$). (F)
- **A** The track slope parameter A ($\mathrm{x} = \mathrm{A} \times \mathrm{z} + \mathrm{B}$). (F)

5.5.5 Blocklet 301

The 301 blocklets store information for preshower clusters which have been matched to a drift chamber track. All information is taken from the DMPS and DOPS banks, except for the z position of the cluster, which is filled with the position of the first reference plane of the preshower, as given by the extrapolator package.

- **PrsIdX** The identifier of the cluster in x. (I)
- **PrsIdY** The identifier of the cluster in y. (I)
- **NTubX** The number of tubes in the cluster in x. (I)
- **NTubY** The number of tubes in the cluster in y. (I)
- **ClusTopo** The cluster topology/overlap word. (I) The following description of the possible values comes from looking at the phase 2 routine **PrsCheckOverlap**.
  - : **0** Both projections are not overlapped.
  - : **1** The x projection is overlapped.

: **2** The y projection is overlapped.

: **3** Both projections are overlapped.

: **4** The x projection is overlapped and the y projection is missing.

: **5** The y projection is overlapped and the x projection is missing.

: **6** One of the overlapped clusters is overlapped in the other projection.

- **XClu** The x position of the cluster. (F)
- **YClu** The y position of the cluster. (F)
- **ZClu** The z position of the cluster. (F)
- **SigCluX** The width of the cluster in x. (F)
- **SigCluY** The width of the cluster in y. (F)
- **ECluH** The energy of the cluster in the horizontal (y) tubes (MIPs). (F)
- **ECluV** The energy of the cluster in the vertical (x) tubes (MIPs). (F)
- **ProbEl** This word stores the probability to be an electron from preshower information. (F) The information which follows on what values it can take comes from looking at the phase 2 function **PrsProbToBeEle**.

  : **-5.0** If it occurs, indicates that supposed matching DC track could not be found.

  : **-1.0** The track does not extrapolate to the preshower at all.

  : **-2.0** Extrapolation is available at the preshower, but the extrapolator "param" structure was unavailable.

  : **-3.0** The extrapolator "param" structure was available, but the extrapolated momentum was $< 1.0 \text{ GeV/c}$.

  : **-4.0** Passed the above cuts, but energy deposited in the preshower was < 0.01 (MIP?).

  : **0.0** The energy released was below a momentum-dependent threshold for electron identification at the specified efficiency.

  : **1.0** The energy released is above the threshold, i.e. this particle is flagged as an electron.

- **PionCon** The pion contamination word. (F) The information which follows on what values it can take comes from looking at the phase 2 function **PrsProbToBeEle**.

  : **-1.0** The track was not flagged as an electron (i.e. it had $\text{ProbEl} \neq 1.0$).

  : > 0.0 Filled with the parametrised integral of the dN/dE spectrum for pions, from threshold to infinity at the current momentum.

5.5.6 Blocklet 304

The 304 blocklets store information for preshower clusters which have been matched to a calorimeter cluster. All information is taken from the DMPS and DOPS banks, except for the z position of the cluster, which is filled with the position of the first reference plane of the preshower, as given by the extrapolator package.

- **PrsIdX** The identifier of the cluster in x. (I)
- **PrsIdY** The identifier of the cluster in y. (I)
- **NTubX** The number of tubes in the cluster in x. (I)
- **NTubY** The number of tubes in the cluster in y. (I)
- **XClu** The x position of the cluster. (F)
- **YClu** The y position of the cluster. (F)
- **ZClu** The z position of the cluster. (F)
- **SigCluX** The width of the cluster in x. (F)
- **SigCluY** The width of the cluster in y. (F)
- **ECluH** The energy of the cluster in the horizontal (y) tubes (MIPs). (F)
- **ECluV** The energy of the cluster in the vertical (x) tubes (MIPs). (F)

5.5.7 Blocklet 401

The 401 blocklets store information for calorimeter clusters which have been matched to a drift chamber track. Information is taken directly from the DMCA and DOCA banks.

Note that in DST v7r1, it was possible to find 401 blocklets which appeared to be empty. This was because all relevant entries in the match table (DMCA banks) were converted to 401 blocklets, whereas in fact some DMCA banks were used only to record extrapolation information indicating that the track in question had reached the calorimeter. At the DST level this information was recorded elsewhere (in blocklet 1201) but the 401 was inadvertently raised in any case. This "feature" no longer appears in the present DST, however it is still possible in a few cases to find 401 blocklets with no measured energy deposition in the calorimeter. In this case, these blocklets exist to indicate that energy is **predicted** to be deposited, and in fact the word **EDepMax** will be found to be filled, giving the predicted upper bound on this energy deposition.

For electron or positron candidates, note that the words **ShChiX**, **ShChiY** and **ShChiBi** contain "raw" $\chi 2$ recalculated using the phase2 function **CalElectronRawChi2**, rather than the "cluster" $\chi 2$ contained in these words in version v7r2.

More information relating to these words may be found by consulting NOMAD memo 97-018 [8] or the phase 2 Web pages [5].

- **CalId** The identifier of the calorimeter cluster. (I)
- **NCells** The number of cells in the cluster. (I)

- **TFlag** Timing flag. (I) The information stored in this word is derived from the bitted calorimeter status word (word 2) of the DOCA bank. The DST word contains six bits, with the following meanings. Note that the bits are labelled from 0 to 31, *not* 1 to 32.
  - **Bit 0** All of the energy in the cluster is out of time.
  - **Bit 1** The cluster contains out of time energy.
  - **Bit 2** The TDC information is **unavailable**.
  - **Bit 3** Out of time flag from match with muon chambers at phase 2.
  - **Bit 4** Out of time flag from match with trigger scintillators at phase 2
  - **Bit 5** This cluster should be ignored.
  - **Bit 6** Was out of time, recovered for phase1 errors (TDC calibration problems) in the first period of 1996. Essentially bits 0 and 1 get untagged.

  It can be seen that a value of zero for the **TFlag** word indicates that all is OK. If it has a non-zero value, then the individual bits will need to be looked at to learn more. Note that if the TDC information is not available (bit 2 set) then **TFlag** will have the value 4, provided bits 4 and 5 are not set. In the previous releases of the DST package, setting of bits 3 and 4 was not possible as the information was not provided at the phase2 level. In the present release, the information is obtained through a DST preprocessing call to two muon library routines, **MUEVENTOTH** and **OTHCLUSTER**.

- **XClu** The x position of the cluster. (F)
- **YClu** The y position of the cluster. (F)
- **ZClu** The z position of the cluster. (F)
- **EClu** The energy of the cluster. (F)
- **Radx** The radius of the cluster in x. (F)
- **Rady** The radius of the cluster in y. (F)
- **EOverP** $E/(P-\sigma_P)$ where E is the energy as measured in the calorimeter and P is the track momentum as given by the drift chambers. Selecting values for this word > 0.85 corresponds to the standard phase2 electron identification cut. Note that in DST v7r1, this word was inappropriately named **ProbEl** since that was what it is called in the phase2 documentation. It has been renamed for this version in order to better reflect its meaning.
- **ESho** The shower energy, corrected for the preshower. (F)
- **ShChiX** The shower profile χ^2 in x (see above note). (F)
- **ShChiY** The shower profile χ^2 in y (see above note). (F)
- **ShChiBi** The global shower profile χ^2 (see above note). (F)
- **EDepMin** The minimum energy deposited in the calorimeter by this track. (F)
- **EDepMax** The maximum energy deposited in the calorimeter by this track. (F)

- **NormEP** The normalized difference between the track momentum and the cluster energy $(E-p)/\sigma(E-p)$. (F)
- **Overlap** The fraction of energy not assigned to the track present in the cells used to build the cluster, i.e. $1.-EClu/Etot$ where $Etot$ is the total energy in the cells associated to track. (F)

5.5.8 Blocklet 404

The 404 blocklets store information for standalone calorimeter clusters. Information is taken directly from the DMCA and DOCA banks.

- **CalId** The identifier of the calorimeter cluster. (I)
- **NCells** The number of cells in the cluster. (I)
- **TFlag** Timing flag. See the description of the corresponding word in the 401 blocklet. (I)
- **GammType** The gamma type. (I) The following description is taken from the Web pages for phase 2 [5].

  : **1** Gamma shower obtained from a cluster with only one local maximum, and **with** associated preshower cluster.

  : **2** Gamma shower obtained from a cluster with only one local maximum, and **without** associated preshower cluster.

  : **3** Gamma shower obtained from a cluster with two local maxima, and **with** associated preshower cluster.

  : **4** Gamma shower obtained from a cluster with two local maxima, and **without** associated preshower cluster.

  : **5** Gamma shower obtained from a cluster with three local maxima, and **with** associated preshower cluster.

  : **6** Gamma shower obtained from a cluster with three local maxima, and **without** associated preshower cluster.

  : **...**

  : **41** Overlap gamma.

  : **42** Consistency gamma.

  : **51** Neutral gamma.

  Overlap gammas are built from neutral preshower clusters (with a signal greater than 2.5 mip). They start as single cell clusters but other energy can be added from the hadron subtraction. Consistency gammas are generated when the electron shower minimization fails.

- **BremTrId** The identifier of the track this photon is associated to, if it occurs in the bremsstrahlung list. (I) The bremsstrahlung algorithm here is the one internal to **calgamma.c**.
- **XClu** The x position of the cluster. (F)
- **YClu** The y position of the cluster. (F)

- **ZClu** The z position of the cluster. (F)
- **EClu** The energy of the cluster. (F)
- **Radx** The radius of the cluster in x. (F)
- **Rady** The radius of the cluster in y. (F)
- **ESho** The shower energy, corrected for the preshower. (F)
- **ShChiX** The shower profile χ^2 in x. (F)
- **ShChiY** The shower profile χ^2 in y. (F)
- **ShChiBi** The global shower profile χ^2. (F)
- **EDep** The energy deposited in the calorimeter by this neutral.
- **ESaved** The energy of the gamma without the bremsstrahlung algorithm applied. (F) The bremsstrahlung algorithm here is the one internal to **calgamma.c**.
- **EDepMin** The minimum energy deposited in the calorimeter by this neutral. (F)
- **EDepMax** The maximum energy deposited in the calorimeter by this neutral. (F)

5.5.9 Blocklet 501

Filling of the 501 blocklet is achieved by extracting information from the DMMU and DOMU banks. The presence of a DMMU bank containing more information than just the muon veto blocklet is the determining factor in whether a 501 blocklet is raised. Since selection of a "good" muon (the **ProbMu** word of the 101 blocklet having value 1.0) has tighter criteria than this, there are more 501 blocklets on the DST than there are "good" muons.

The algorithm currently used to match up the information in the DMMU and DOMU banks when filling the blocklet is the following. A loop is made through the DMMU bank, skipping muon veto blocklets, selecting blocklets which correspond to TKU objects, tracks in space. For each selected object, a loop is then performed over the objects in the DOMU bank to find the blocklet with the matching identifier. Information is then extracted from the subblocklets (corresponding to TUPs, tracks in projection) for this blocklet. A second loop over DMMU blocklets, without the requirement that the object be a track in space, is performed to pick up those cases where only a single projection has been used in the match. It should be noted that this procedure does not guarantee that the identical TUPs to those used in the matching process are obtained when the match used tracks in projection from different modules, as may be the case in the overlap region, for example. The algorithm may be replaced by a better one in future versions of the package.

- **MuoIdS1** This is a packed integer word giving the identifiers of the TUPs corresponding to the x and y tracks in projection in station 1. (I) If the TUP identifiers are ITUPX1 and ITUPY1, then $\mathrm{MuoIdS1} = 100000 \times \mathrm{ITUPX1} + \mathrm{ITUPY1}$
- **MuoIdS2** The same as **MuoIdS1** only for station 2. (I)
- **QualS1** Decision of "old" phase 2 (χ^2) on muon matching, station 1. (F)

- **QualS2** Decision of "old" phase 2 (χ^2) on muon matching, station 2. (F)
- **Chi2S1** The χ^2 in space for station 1 is filled in the following way. The χ^2/NDF for this station is obtained from the DMMU bank following the selection criterion described above. The NDF is set to 2 or 4 depending on whether one or both projections (TUPs) are present for the match. The two quantities are multiplied together to give the χ^2. On the DST the number of degrees of freedom can be deduced from whether of not both identifiers are present in the **MuoId** word for the station. (F)
- **Chi2S2** The same as **Chi2S1** only for station 2. (F)
- **XS1** The x position of the muon track in station 1. This and the following words are taken from the relevant subblocklet of the DOMU bank. (F)
- **YS1** The y position of the muon track in station 1. (F)
- **ZS1** The z position of the muon track in station 1. (F)
- **XS2** The x position of the muon track in station 2. (F)
- **YS2** The y position of the muon track in station 2. (F)
- **ZS2** The z posiiton of the muon track in station 2. (F)
- **SlXS1** The slope in x of the muon track in station 1. (F)
- **SlYS1** The slope in y of the muon track in station 1. (F)
- **SlXS2** The slope in x of the muon track in station 2. (F)
- **SlYS2** The slope in y of the muon track in station 2. (F)
- **MuonT0** This word is reserved for the t_0 determined for this match. It is currently not available : the word is filled with 0.0 (as in the phase2 bank). (F)
- **MuPest** Muon momentum estimate (from comparison of muon tracks in station 1 and 2). This is filled by calling the muon code access routine **MUMOMEST**. (F)

5.5.10 Blocklet 505

The 505 blocklet contains information on standalone muon tracks. Specifically, the DOMU banks are scanned, looking for TKU objects (tracks in space). Each TKU object becomes an entry in the 505 blocklet, which is of variable length.

The first word in the blocklet gives the number of objects stored.

- **NEnt** The number of muon standalone tracks (TKU objects). (I)

For each of the **NEnt** tracks, the following is stored.

- **TrkId** The identifier of the muon track. This is a 4 digit number, with the most significant digit giving the module in which this track occurs (1 to 5). (Note that previous versions of this documentation erroneously indicated that it was the station that was given). (I)

- **X** The x position of the muon track. (F)
- **Y** The y position of the muon track. (F)
- **Z** The z position of the muon track. (F)
- **SlX** The slope in x of the muon track. (F)
- **SlY** The slope in y of the muon track. (F)
- **MuonT0** The t_0 for the muon track. Note that this word is only non-zero for those tracks which have been flagged as out of time. (F)

5.5.11 Blocklet 701

The 701 blocklets store information for hits in the veto scintillators which have been matched to a drift chamber track.

- **VetId** Identifier of the VET object. (I)
- **NHits** The number of in-time hit counters. (I)

5.5.12 Blocklet 707

The 707 blocklets store information for in-time hits in the veto scintillators which have not been matched to a drift chamber track.

- **VetId** Identifier of the VET object. (I)
- **NHits** The number of in-time hit counters. (I)

5.5.13 Blocklet 801

The 801 blocklets store information for matches of drift chamber tracks, extrapolated back to the front calorimeter, with activity in FCAL.

- **FcaId** The identifier of the FCAL object. (I)
- **XExt** The x position of the extrapolated track at the FCAL. (F)
- **YExt** The y position of the extrapolated track at the FCAL. (F)
- **SigXExt** The error σ_x on the x position of the extrapolated track at the FCAL. (F)
- **SigYExt** The error σ_y on the y position of the extrapolated track at the FCAL. (F)
- **TZero** The t_0 of the track at the z position of the FCAL vertex. (F)
- **Chi2** The association χ^2 for the match. (F)

5.5.14 Blocklet 808

The 808 blocklet stores standalone information for the front calorimeter. The blocklet is variable in length, and will appear at most once per event.
The fixed part of the blocklet contains the following information.

- **FcaId** The identifier of the FCAL object (0 for now). (I)
- **NEnt** The number of modules for which information is stored in this blocklet. (I)
- **VType** Flags whether the vertex determination came from the drift chambers of FCAL. (I)
  : **1** Drift chamber information.
  : **8** FCAL information.
- **XPri** The x position of the primary vertex in the FCAL. (F)
- **YPri** The y position of the primary vertex in the FCAL. (F)
- **ZPri** The z position of the primary vertex in the FCAL. (F)
- **MipToGeV** The Mip to GeV value used for this event. (F)

For each of the **NEnt** modules contributing to this blocklet, the following information is stored.

- **ModId** The FCAL module identifier. (I)
- **EDep** The energy deposited in the module. (F)
- **X** The x position in the module. (F)
- **Y** The y position in the module. (F)
- **Z** The z position in the module. (F)
- **SigX** The error σ_x on the x position in the module. This word is not yet filled, and will be found to be identically -999.0. (F)
- **TZMin** The minimum possible t_0 for this module. (F)
- **TZMax** The maximum possible t_0 for this module. (F)

5.5.15 Blocklet 901

The 901 blocklets store information for HCAL clusters which have been matched to a drift chamber track. All of the information is derived from the phase 2 DMHC and DOHC banks, which the exception of the z position of the HCAL cluster, for which the z position of the first HCAL reference plane as given by the extrapolator package is stored, and the energy corrected for non-linear effects, as explained below.

- **HcaId** The identifier of the HCAL cluster. (I)
- **XClu** The x position of the HCAL cluster. (F)

- **YClu** The y position of the HCAL cluster. (F)
- **ZClu** The z position of the HCAL cluster. (F)
- **EClu** The energy of HCAL cluster. (F)
- **ECorr** The energy deposited in ECAL and HCAL corrected for non-linearities. As the information is not present in the current phase2 output, it has been obtained by calling a phase2 library routine **HCAL˙V7R2˙PATCH** provided by P. Hurst. (F) See [9] for a discussion of the correction method.

5.5.16 Blocklet 904

The 904 blocklets store information for HCAL clusters which have been matched to a standalone calorimeter cluster. All of the information is derived from the phase 2 DMHC and DOHC banks, which the exception of the z position of the HCAL cluster, for which the z position of the first HCAL reference plane as given by the extrapolator package is stored, and the energy corrected for non-linear effects, as explained below.

- **HcaId** The identifier of the HCAL cluster. (I)
- **XClu** The x position of the HCAL cluster. (F)
- **YClu** The y position of the HCAL cluster. (F)
- **ZClu** The z position of the HCAL cluster. (F)
- **EClu** The energy of HCAL cluster. (F)
- **ECorr** The energy deposited in ECAL and HCAL corrected for non-linearities. As the information is not present in the current phase2 output, it has been obtained by calling a phase2 library routine **HCAL˙V7R2˙PATCH** provided by P. Hurst. (F) See [9] for a discussion of the correction method.

5.5.17 Blocklet 909

The 909 blocklets store information for standalone HCAL clusters. All of the information is derived from the phase 2 DMHC and DOHC banks, which the exception of the z position of the HCAL cluster, for which the z position of the first HCAL reference plane as given by the extrapolator package is stored, and the energy corrected for non-linear effects, as explained below.

- **HcaId** The identifier of the HCAL cluster. (I)
- **XClu** The x position of the HCAL cluster. (F)
- **YClu** The y position of the HCAL cluster. (F)
- **ZClu** The z position of the HCAL cluster. (F)
- **EClu** The energy of HCAL cluster. (F)
- **ECorr** The energy deposited in ECAL and HCAL corrected for non-linearities. As the information is not present in the current phase2 output, it has been obtained by calling a phase2 library routine **HCAL˙V7R2˙PATCH** provided by P. Hurst. (F) See [9] for a discussion of the correction method.

5.5.18 Blocklet 1101

The 1101 blocklet stores information on calorimeter clusters associated with drift chamber tracks using the alternative clustering algorithm of Gary Feldman. The information is derived from the DEHC bank at phase 2.

Included in this blocklet is information obtained from the bremsstrahlung strip algorithm of Gary Feldman (as opposed to the Padova algorithm - see the 1301 blocklet). If **NEnt** is non-zero, this track has been identified as having bremsstrahlunged, and the blocklet will contain a list of identifiers of objects contributing to the bremsstrahlung strip. Thus this blocklet is in general of variable length.

- **FelId** The identifier of the cluster. (I)
- **NCells** The number of cells in the cluster. (I)
- **Type** The cluster type. (I) Valid cluster types are:
  - : **1** charged hadron.
  - : **4** electron.
  - : **5** muon.
- **TrkId** The identifier of the associated track (if this cluster has been included in the bremsstrahlung strip). (I)
- **NEnt** The number of objects contributing to the bremsstrahlung strip. (I)
- **NTrk** The number of tracks contributing to the bremsstrahlung strip. By subtraction of this word from **NEnt** the number of clusters contributing is obtained. (I)
- **XClu** The x position of the calorimeter cluster. (F)
- **YClu** The y position of the calorimeter cluster. (F)
- **ZClu** The z position of the calorimeter cluster. (F)
- **EClu** The energy of the calorimeter cluster. (F)
- **Radx** The radius of the calorimeter cluster in x. (F)
- **Rady** The radius of the calorimeter cluster in y. (F)
- **BremB** The bremsstrahlung strip energy. (F)

The variable part of the blocklet is a list of identifiers of objects forming the bremmstrahlung strip for this track.

- **ObjId** The identifier of the object used in the bremsstrahlung strip sum. If charged (i.e. the first NTrk objects) this is a **DchId** as stored in the 101 blocklets. If neutral (i.e. the rest of the entries in the list) this is a **FelId** as stored in the 1111 blocklets. (I)

5.5.19 Blocklet 1104

The 1104 blocklet stores information on neutral calorimeter clusters formed using the alternative clustering algorithm of Gary Feldman, which have been matched to standalone HCAL clusters. Note that these HCAL clusters have *in turn* been matched within the standard matching engine framework to "standard" neutral calorimeter clusters - if this were not the case these blocklets would be labelled 109. Thus, the blocklet ID of 1104 does not constitute a direct match between calorimeter clusters from the two clustering algorithms, but rather an indirect match due to both clusters matching to the same standalone HCAL cluster.

Note that situation here, like in DST version v7r3, is *quite different* to that in DST version v7r2. In that case, the 1104 blocklets represented standalone calorimeter clusters (i.e. not matched to a drift chamber tracks). In this version such clusters are labelled 1111, which better reflects their origin. The change has been necessitated by the introduction of matched between Feldman ECAL clusters and neutral HCAL clusters.

- **FelId** The identifier of the calorimeter cluster. (I)
- **NCells** The number of cells in the cluster. (I)
- **TrkId** The identifier of the associated track (if this cluster has been included in the bremsstrahlung strip). (I)
- **XClu** The x position of the calorimeter cluster. (F)
- **YClu** The y position of the calorimeter cluster. (F)
- **ZClu** The z position of the calorimeter cluster. (F)
- **EClu** The energy of the calorimeter cluster. (F)
- **Radx** The radius of the calorimeter cluster in x. (F)
- **Rady** The radius of the calorimeter cluster in y. (F)

5.5.20 Blocklet 1109

The 1109 blocklet stores information on neutral calorimeter clusters formed using the alternative clustering algorithm of Gary Feldman, which have been matched to standalone HCAL clusters.

- **FelId** The identifier of the calorimeter cluster. (I)
- **NCells** The number of cells in the cluster. (I)
- **TrkId** The identifier of the associated track (if this cluster has been included in the bremsstrahlung strip). (I)
- **XClu** The x position of the calorimeter cluster. (F)
- **YClu** The y position of the calorimeter cluster. (F)
- **ZClu** The z position of the calorimeter cluster. (F)

- **EClu** The energy of the calorimeter cluster. (F)
- **Radx** The radius of the calorimeter cluster in x. (F)
- **Rady** The radius of the calorimeter cluster in y. (F)

5.5.21 Blocklet 1111

The 1111 blocklet stores information on standalone calorimeter clusters formed using the alternative clustering algorithm of Gary Feldman. The information is derived from the DEHC bank at phase 2. *Note that in previous DST versions, this blocklet had ID 1104. It has been changed to be more consistent, and to fit in with new blocklets relating to HCAL matching to Gary Feldman calorimeter clusters.*

Each cluster will be found in a separate 1111 blocklet. Note that a cut of 30 MeV on cluster energy has been applied in selecting clusters for inclusion in the DST. One should also note that although such a cut cluster will not appear on the DST, it may still appear in the list of identifiers for the bremsstrahlung strip as stored in an 1101 blocklet, if it contributed to the strip.

- **FelId** The identifier of the calorimeter cluster. (I)
- **NCells** The number of cells in the cluster. (I)
- **TrkId** The identifier of the associated track (if this cluster has been included in the bremsstrahlung strip). (I)
- **XClu** The x position of the calorimeter cluster. (F)
- **YClu** The y position of the calorimeter cluster. (F)
- **ZClu** The z position of the calorimeter cluster. (F)
- **EClu** The energy of the calorimeter cluster. (F)
- **Radx** The radius of the calorimeter cluster in x. (F)
- **Rady** The radius of the calorimeter cluster in y. (F)

5.5.22 Blocklet 1201

The 1201 blocklet stores extrapolator information for drift chamber tracks. The information is extracted from the TXD banks, and is stored at a single reference plane (the first) for the preshower, calorimeter and HCAL, and at two reference planes (one for each station) for the muon chambers. A blocklet is present only if there is information for at least one of these planes. If the blocklet exists but a particular reference plane was not reached, the words for that plane will be filled with zeros.

In the TXD banks, the momentum information is stored as $1/p$ and the slopes of the track. To be consistent with the storage of momenta elsewhere on the DST, this information has been converted to (p_x, p_y, p_z) using the routine **track˙cartesian** from the extrapolator package.

Note that at present, no errors on the extrapolated quantities are stored on the DST. The information which is stored for each extrapolated track is the following.

- **XPrs** The x position at the PRS reference plane. (F)

- **YPrs** The y position at the PRS reference plane. (F)
- **ZPrs** The z position at the PRS reference plane. (F)
- **PxPrs** The x component of momentum at the PRS reference plane. (F)
- **PyPrs** The y component of momentum at the PRS reference plane. (F)
- **PzPrs** The z component of momentum at the PRS reference plane. (F)
- **XCal** The x position at the CAL reference plane. (F)
- **YCal** The y position at the CAL reference plane. (F)
- **ZCal** The z position at the CAL reference plane. (F)
- **PxCal** The x component of momentum at the CAL reference plane. (F)
- **PyCal** The y component of momentum at the CAL reference plane. (F)
- **PzCal** The z component of momentum at the CAL reference plane. (F)
- **XHca** The x position at the HCA reference plane. (F)
- **YHca** The y position at the HCA reference plane. (F)
- **ZHca** The z position at the HCA reference plane. (F)
- **PxHca** The x component of momentum at the HCA reference plane. (F)
- **PyHca** The y component of momentum at the HCA reference plane. (F)
- **PzHca** The z component of momentum at the HCA reference plane. (F)
- **XMuoS1** The x position of the extrapolated track at muon station 1. (F)
- **YMuoS1** The y position of the extrapolated track at muon station 1. (F)
- **ZMuoS1** The z position of the extrapolated track at muon station 1. (F)
- **PxMuoS1** The x of the extrapolated track at muon station 1. (F)
- **PyMuoS1** The y of the extrapolated track at muon station 1. (F)
- **PzMuoS1** The z of the extrapolated track at muon station 1. (F)
- **XMuoS2** The x position of the extrapolated track at muon station 2. (F)
- **YMuoS2** The y position of the extrapolated track at muon station 2. (F)
- **ZMuoS2** The z position of the extrapolated track at muon station 2. (F)
- **PxMuoS2** The x of the extrapolated track at muon station 2. (F)
- **PyMuoS2** The y of the extrapolated track at muon station 2. (F)

- **PzMuoS2** The z of the extrapolated track at muon station 2. (F)

There is one “feature” of the TXD banks (and hence the 1201 blocklet) which users should note. It appears that in those cases where a track has been identified as an electron, the track (or rather the “shower” resulting from the track) has been extrapolated as a neutral through to the muon chambers. Thus, if the word **ProbEl** in the 201 blocklet has the value 1.0, the 1201 blocklet will be found to contain non-zero extrapolation information for all subdetectors (even though the electron itself will not normally get that far), and further, the 3-momentum of the extrapolated track will not change from subdetector to subdetector (since the extrapolation was as a neutral). Due caution should be exercised in interpreting the contents of the blocklet in these cases.

5.5.23 Blocklet 1301

The 1301 blocklet stores information from the Padova bremsstrahlung strip algorithm. It is an **exact** copy of the DMBR bank which is raised in phase 2 processing, and is variable in length since it contains a list of cross-referencing information to the objects making up the bremsstrahlung strip. Note that this version of the DMBR bank (and hence this blocklet) contains in addition to the list of bremsstrahlung objects a list of information for calorimeter cells relevant to the strip.

The fixed part of the blocklet has the following format.

- **NHead** Header length in words. (I)
- **Version** Blocklet format version number. (I)
- **NEnt** The number of objects stored in blocklet. This includes both the list of bremsstrahlung objects and the cell list. (I)
- **NBrem** The number of bremsstrahlung objects stored in the blocklet. By subtracting **NBrem** from **NEnt** the number of objects in the cell list is obtained. (I)
- **EType** The electron track type. (I) Valid values are:
  - : **0** Unknown.
  - : **1** Single track.
  - : **2** Broken track.
- **EBremss** The total bremsstrahlung energy. (F)
- **SEBremss** The error on the total bremsstrahlung energy. (F)
- **EGamma** The total gamma energy. (F)
- **EPrimary** The primary nonet energy. (F)
- **EPhoton** The total photon energy. (F)
- **EPrsX** The total x signal for the preshower. (F)
- **EPrsY** The total y signal for the preshower. (F)
- **X0Pass** The amount of X_0 passed. (F)

- **LTrack** The total charged track length (cm). (F)
- **ELostTk** Total track energy not accounted for in the list. This will be found to be identically 0.0 at the present time. (F)
- **ELostCal** Total calorimeter energy not accounted for in the list. This will be found to be identically 0.0 at the present time. (F)
- **EMC** The generated energy (Monte Carlo). (F)

The variable part of the blocklet is in two parts, the bremsstrahlung list of cross-references, and the list of cell information. The first **NBrem** entries give the bremsstrahlung list.

- **BitVal** Bitted word containing bremsstrahlung list. (I)
  - **Bit 00** If set to 1, this drift chamber track is inside the bremsstrahlung list.
  - **Bit 01** If set to 1, this preshower cluster in x is inside the bremsstrahlung list.
  - **Bit 02** If set to 1, this preshower cluster in y is inside the bremsstrahlung list.
  - **Bits 03 - 10** The identifier of the track/cluster. Note that if bits 00 to 02 are all zero, the Id will be that of a calorimeter cluster (the Calid word of a 401/404 blocklet). If bit 00 is set, the Id will be that of a drift chamber track (the DchId word of a 101 blocklet).
  - **Bits 11 - 20** The fraction of the calorimeter cluster inside the bremsstrahlung list $(\times 1000)$
  - **Bits 21 - 25** The bremsstrahlung shower track type.
    - **1** Electron at primary
    - **2** Electron that triggered the brem call
    - **3** 1 + 2 (electron at primary + triggering electron)
    - **4** Electron tail
    - **5** Gamma prong overlapping electron nonet
    - **6** Gamma prong (ECAL nonet used)
    - **7** Gamma prong (DC momentum used)
    - **8** Asymmetric gamma overlapping electron nonet
    - **9** Asymmetric gamma (ECAL nonet used)
    - **10** Asymmetric gamma (DC momentum used)
    - **11-15** Not used
    - **16** Middle track
    - **As before +16** Tail of broken track

The last $\mathbf{NEnt} - \mathbf{NBrem}$ entries give the cell list.

- **BitVal** Bitted word containing cell list. (I)
  - **Bits 00 - 11** The identifier of the cell $(\mathrm{column} \times 100 + \mathrm{row})$.
  - **Bits 12 - 31** The energy deposited in the cell (MeV).

Note that a bug at the phase2 (v7r3c) level in filling of the cell list in the DMBR bank has caused the information in the cell list (only) for the v7r2 DST to be incorrect. For the present release, the correct cell list has been generated by calling at the DST preprocessor level a phase2 library routine **CheckDMBR** provided by S. Lacaprara and M. Contalbrigo which regenerates the correct DMBR bank from phase2 information.
More information on the use of the Padova bremsstrahlung strip can be found by consulting the phase2 Web pages [5].

5.6 The LEPS bank

The LEPS bank summarises the generated interaction produced by NEGLIB. It is essentially a copy of the information available in the LUJT and LEPT banks from the full NOMAD ZEBRA structure. The bank consists of a header part and a repeat part which is a table of generated particles. The header contains the following information. All words except **NPart** and **IdPar** come from the LEPT bank.

- **NPart** This is the number of particles contained in the LUJET common for this event, as stored in the LUJT bank. (I)
- **IdPar** The identifier of the parent particle to the neutrino (as determined by which of the NUBEAM tables: pion, kaon, other was used) which decayed to produce the intercating neutrino. The code stored is the GEANT particle code. The word is a copy of word 14 of the SEVT bank. (I)
- **Enu** The incoming neutrino energy. (F)
- **Xbj** Bjorken x for the event. (F)
- **Ybj** Bjorken y for the event. (F)
- **Wsq** W^2 for the event. (F)
- **Qsq** Q^2 for the event. (F)
- **Nu** ν for the event. (F)

The repeat part of the bank contains the following information for each of the **NPart** particles in the LUJET common.

- **Type** Particle type (10000*K(PART,1)+K(PART,3)). (I) K(PART,1) is the status (KS) code of the present parton/particle, giving information on for example its state of decayedness. K(PART,3) gives information on where to find the parent particle or jet that produced this particle. For more details see reference [10], section 5.2.
- **Code** Particle code, (K(PART,2)). (I) This is the so-called parton/particle KF code adopted by JETSET and the Particle Data Group. For the details, see reference [10], section 5.1.
- **Px** The x component of the particle momentum. (F)
- **Py** The y component of the particle momentum. (F)
- **Pz** The z component of the particle momentum. (F)
- **E** The energy of the particle. (F)
- **Mass** The mass of the particle. (F)

5.7 The SVXS bank

The SVXS bank is essentially a summary in tabular form of the information in the SVTX chain of banks. A crude filtering of the vertices is performed in order to reduce the size of the bank. All end vertices (vertices of type 10) are suppressed, as well as vertices whose z position lies beyond the reference plane of HCAL as defined in the extrapolator package. End point vertices of muon tracks are retained in all cases.

- **NVert** Indicates how many vertices are stored in the bank. (I)

For each vertex, the following words are stored.

- **Id** The vertex identifier. Since the simulated vertex structure does not define an explicit ID for the individual vertices, this identifier is filled on the DST by recording the position of the corresponding SVTX bank in the linear chain. (I)
- **Type** The vertex type. Valid types are in principle the same as given in the description of the RVXS bank - note however that not all valid types appear to be used in the corresponding word in the SVTX bank which is simply copied into the present word. (I)
- **NChgd** Counts all charged tracks associated to the vertex (by looping over the STRK structure associated with the simulated vertex, checking on charge). (I)
- **NNeut** Counts all neutral tracks associated to the vertex (by looping over the STRK structure associated with the simulated vertex, checking on charge). (I)
- **X** The x position of the vertex. (F)
- **Y** The y position of the vertex. (F)
- **Z** The z position of the vertex. (F)

5.8 The STKS bank

In a similar way to the SVXS bank, the STKS bank is a summary in tabular form of the information in the STRK banks. A crude filtering is performed in order to reduce the size of the bank. All tracks with both beginning and end point vertices beyond the preshower, or with momentum at the beginning vertex less than 30 MeV are ignored. Muon tracks are always retained.

- **NTrak** Indicates how many tracks are stored in the bank. (I)

For each track, the following words are stored.

- **Id** The track identifier. Since the simulated track bank does not define an explicit ID for the individual tracks, this identifier is filled on the DST by recording the position of the STRK bank in the list obtained by looping over all tracks from each vertex in the SVTX/STRK structure. (I)
- **LId** The LEPTO identifier of the track. In practice this works out to be the **index**, or row number, of the track in the table given in the LEPS bank. It will be zero for all tracks not in the LEPS bank (i.e. not forming part of the particle list generated by LEPTO). In most places on the DST such a word would be called **Indx....** rather than **Id**; the fact that the identifier acts as an index in this case is fortuitous. (I)

- **GId** The Geant identifier of the track. This can in principle be cross-referenced with the **Id** word in the SCAS bank. Otherwise it is simply an index into the list of all simulated tracks in the event. Since not all simulated tracks are stored on the DST, this is only of use when going back to the event at the GENOM level or on the full RECON output. (I)
- **IndxVtxB** This word gives the index (i.e. the row in the table of all vertices) of the beginning vertex of this track in the SVXS bank. It can therefore be used directly as the second argument to any of the access functions returning individual words from a particular vertex in the SVXS bank. Note that in the printout of the event given by the **PrintDST()** routine, both this index and the actual ID of the vertex which has been indexed are printed. The latter is called **VIdB** and is printed in brackets, to indicate that it is not actually stored in the bank. Printing this is to aid those users more likely think in terms of the ID of the vertex than in terms of its row position in a table. It is important to understand the distinction when using this word. (I)
- **IndxVtxE** The same as the previous word only this time for the end vertex. Note that when the end vertex has not been included in the SVXS bank, this word will be zero. (I)
- **Charge** The charge of the track. Since the charge is not stored in the STRK bank, it is deduced from the GEANT code described next. (I)
- **Type** The GEANT code of the track. **Important Note:** In general the code stored in the STRK bank from which this word is taken follows the GEANT scheme for numbering particles. In the case of tau leptons, NOMAD has adopted the policy of creating a short track and secondary vertex for the tau, and the code stored here will found to be 100 (τ^-) or 101 (τ^+). In the case of some charmed particles, for which a short track and secondary vertex is also created, there is no GEANT code, and yet no NOMAD specific numbering convention has been defined. In these cases, what will be found in this word is actually the Particle Data Group (used by LEPTO, JETSET) code. This mixing of conventions can be confusing unless the user is aware of it. (I)
- **PxB** The x component of the track momentum at the beginning vertex. (F)
- **PyB** The y component of the track momentum at the beginning vertex. (F)
- **PzB** The z component of the track momentum at the beginning vertex. (F)
- **PxF** The x component of the track momentum at the first hit (will be 0.0 for neutral tracks). (F)
- **PyF** The y component of the track momentum at the first hit (will be 0.0 for neutral tracks). (F)
- **PzF** The z component of the track momentum at the first hit (will be 0.0 for neutral tracks). (F)
- **PxL** The x component of the track momentum at the last hit (will be 0.0 for neutral tracks). (F)
- **PyL** The y component of the track momentum at the last hit (will be 0.0 for neutral tracks). (F)
- **PzL** The z component of the track momentum at the last hit (will be 0.0 for neutral tracks). (F)
- **PxE** The x component of the track momentum at the end vertex (will be 0.0 if the end vertex is not included in the SVXS bank). (F)
- **PyE** The y component of the track momentum at the end vertex (will be 0.0 if the end vertex is not included in the SVXS bank). (F)

- **PzE** The z component of the track momentum at the end vertex (will be 0.0 if the end vertex is not included in the SVXS bank). (F)
- **XF** The x position of the first hit on the track (will be 0.0 for neutral tracks). (F)
- **YF** The y position of the first hit on the track (will be 0.0 for neutral tracks). (F)
- **ZF** The z position of the first hit on the track (will be 0.0 for neutral tracks). (F)
- **XL** The x position of the last hit on the track (will be 0.0 for neutral tracks). (F)
- **YL** The y position of the last hit on the track (will be 0.0 for neutral tracks). (F)
- **ZL** The z position of the last hit on the track (will be 0.0 for neutral tracks). (F)
- **Length** The track length (in cm). Note an important change from DST version v7r2. In v7r2, since the STKS bank was filled from the C structures into which the SVTX/STRK banks were unpacked by the **dc** package, and no function to return the track length was available, a length was calculated (in the DST code) using the position of the beginning and end vertices. In some cases (especially for long charged tracks with segments outside of the magnetic field, such as muons) the algorithm used was too crude to give a reasonable result, and so the DST value could differ substantially from that stored in the STRK bank. In the present DST version, the length stored is that taken **directly** from the STRK bank. (F)

The STKS bank is the only instance in the current DST library where the information used to fill the bank does not come **entirely** from the corresponding ZEBRA bank in the phase2 output structure. Since (a) some of the information stored in the STKS bank is not stored in the STRK bank itself, (b) the simulated vertices and tracks are unpacked during the running of RECON into C structures and then left unmanipulated, and (c) the utilities to access the C structures are more convenient than attempting to obtain the information directly from the ZEBRA banks in this case, the C structures have been accessed to obtain the extra information in filling the banks, specifically the **VxtrkList** chained list of structures is used. (Note that in DST v7r2, the C structures were used for *all* the information).

5.9 The SCAS bank

The SCAS bank gives a list of the simulated calorimeter information for a Monte Carlo event, at the level of energy deposition in individual lead glass blocks. The contents are taken without further selection from the RCAL bank produced by GENOM.

- **NCells** Indicates how many “cell hits” there are stored in the bank. If several tracks deposited energy in a given lead glass cell, there will be several entries for that cell contributing to **NCells** i.e. the word does not count the number of cells containing deposited energy. (I)

For each cell hit, the following words are stored.

- **Id** The identifier stored here is that assigned to the simulated track depositing the energy by the dc package, and returned by the function **DcTrackGeant(˚DcTrack dctrack)**. It can in principle be cross-referenced with the **GId** word in the STKS bank. (I)

- **LId** The LEPTO identifier stored here is the identifier assigned to the simulated track depositing the energy by the dc package, and returned by the function **DcTrackLepto(˚DcTrack dctrack)**. In practice this works out to be the **index**, or row number, of the track in the table given in the LEPS bank. It will be zero for all tracks not in the LEPS bank (i.e. not forming part of the particle list generated by LEPTO). In most places on the DST such a word would be called **Indx....** rather than **Id**; the fact that the identifier acts as an index in this case is fortuitous. (I)
- **Type** The GEANT code of the track depositing the energy. (I)
- **BlockId** Calorimeter block identifier. (I) ($100 \times \mathrm{Column} + \mathrm{Row}$)
- **E** The energy deposited for this hit. (F)
- **EC** The energy deposited (Čerenkov weighted) for this hit. (F)
- **XAv** The average x position (relative to cell centre). (F)
- **YAv** The average y position (relative to cell centre). (F)
- **ZAv** The average z position (relative to cell centre). (F)

Note that in the above description, the **x**, **y** and **z** information stored for a hit is **relative** to the centre of the cell, that is, it is not an absolute position in the NOMAD coordinate system but rather an offset with respect to the cell centre. This simply reflects the way the information is stored in the RCAL bank.

Chapter 6: Code

In this chapter a brief description of the source code organisation of the DST package is given, to aid anyone needing to develop or understand the code in some detail. Those readers not requiring such detailed knowledge can safely skip this chapter.

The package has been placed under CVS on the NOMAD offline cluster, and can be accessed using the standard “cvs checkout dst”. The directory structure under the **dst** directory is organised into an **include** subdirectory containing .h and .inc files, and a **src** subdirectory in which the majority of the code is placed. The subdirectory denoted **phase2** contains some routines which are used to fill information which cannot be obtained directly from the phase2 output ZEBRA structure, since phase2 has not been repassed. These routines should rightly reside in libraries which create the phase2 output structure, in subsequent phase2 and other library versions. Documentation files are stored in the subdirectory **doc**. Subdirectory **scripts** contains awk and Bourne shell scripts related to automatic generation of DST code. Finally, subdirectory **histos** contains some code to enable histogramming of the individual words stored on the DST.

The source code is a mixture of FORTRAN and C. The filling of the DST ZEBRA banks from the phase2 banks is realised in FORTRAN. C is used for the majority of the access routines, utilising the C structures onto which the ZEBRA banks are mapped.

6.1 Include files

The following include files are relevant to the DST package.

- **dstcom.inc** This FORTRAN include file is used internally by the DST package, when creating the bank structure. It in turn includes the file **dstparams.inc** described next.
- **dstparams.inc** This FORTRAN include file holds a set of parameters defining the bank structure of the DST. It defines the DST version, and defines a set of parameters specifying the number of words in each bank. All routines involved with raising the banks work from these parameters, and they may also be useful for FORTRAN direct ZEBRA bank navigators. This file is automatically generated from the bankdoc using the awk script **dstparams.awk**.
- **dstlink.inc** A further FORTRAN include file which contains the common for the temporary link area used for storing links during raising of banks.
- **dststats.inc** Contains a common block used to store statistics on the size of the DST (these statistics are compiled by routine **DSTSIZE**, called by **CreateDST()**), in order to provide a summary which is printed by **DSTEND**.
- **dstaccess.inc** The common allowing access by name for FORTRAN high-level access. See the documentation for Fergus Wilson’s access package [2] for more details.
- **dstgen.inc** This include file contains a declaration of the type (INTEGER or REAL) of all automatically generated single word access functions. It may be of use to those FORTRAN programmers who use such access functions, in that inclusion of this file in the source code will cover the declaration of any functions used. Two points should be noted, however:
  – The include file **pointer.inc** must be included along with this include file, to make available the POINTER type which is used to declare the navigation routines.

  – The use of **dstgen.inc** will cause many compiler warning messages (one for each function **not** used) if the code is compiled in the standard way. These warnings indicate that a function or variable has been declared but not used. There are ways to avoid these warning messages – for example on the alphas the **f77** compiler option **-warn nouncalled** could be used. This will of course have the side effect of suppressing the ability to spot such warnings for other parts of the code.
  – Finally, a number of auxilliary include files internal to the functioning of DST Fortran code used for extracting information from the phase2 ZEBRA structure are listed here for completeness. **comtrdhl.inc**, **dstdmca.inc**, **dstdmfc.inc**, **dstdmhc.inc**, **dstdmmu.inc**, **dstdmps.in** **dstdmtr.inc**, **dstdmve.inc**, **dstfcamods.inc**, **dstfelbremobj.inc**, **dstmutrk.inc**, **dstmuveto.inc** **dsttxd.inc**
- **muends.inc** List of pointers to muon end vertices in SVTX bank.
- **dstbanks.h** Contains the definition of C structures which map to the ZEBRA DST banks. This file is automatically generated from the bankdoc using the awk script **dstbanks.awk**.
- **dstgen.h** Contains prototypes for the C access routines to the DST banks. This file is automatically generated from the bankdoc using the awk script **dstgen.awk**.
- **dstlepto.h** Prototypes utility function for use with the LEPTO information.
- **dstcfortran.h** Dummied. This redundant include file contained in previous library versions the cfortran interface for calling of DST FORTRAN routines from C code. In the present version these calls have been stored with the routines themselves.
- **utilfilldst.h** Prototypes for some utility C routines used internally by the package for filling of the DST banks. In a sense, these routines are private, which means that they may change without warning between releases of the code. Users may use these routines if they so wish, of course, but should be prepared for surprises.
- **utilaccessdst.h** Prototypes for some utility C routines used publicly for accessing the DST.

6.2 The Source Code

The structure of the DST package will be briefly described in this section. The source code will be found in the subdirectory **src**. All FORTRAN routines in the package will be found in their own separate **.F** file, with the basename being the routine name in lowercase. For example, the routine **DSTEND** will be found in the file **dstend.F**. C functions are generally collected together into appropriate **.c** files, and the name of the file containing a given function will be indicated when the function is described.

6.2.1 Creation of the DST structure

Creation of a DST structure involves calling of the FORTRAN routine **CreateDST**, which may be found in the file **src/createdst.F**. As indicated above, generally any FORTRAN routine referred to below will be found in its own **.F** file in this directory.

CreateDST is essentially a loop over all of the different banks comprising the DST structure, booking and filling. Specifically, the following FORTRAN routines are called:

→ DSTINI to perform package initialization.

→ DstPreProcess to obtain any necessary information which is not present in the phase2 ZEBRA structure, due to the fact that phase2 is not repassed to create v7r4 DSTs.

→ CRDST to create the DST bank.

→ CREVS to create the EVS bank.

→ CRRVXS to create the RVXS bank.

→ CRLEPS to create the LEPS bank.

→ CRSVXS to create the SVXS bank.

→ CRSTKS to create the STKS bank.

→ CRSCAS to create the SCAS bank.

→ CRMAS to create the chain of MAS banks.

→ FILLEVS to fill the EVS bank.

→ DSTSIZE to compile statistics on bank sizes.

The preprocessor routine, **DstPreProcess**, calls the following routines:

→ CHECKDMBR (from phase2 library) to correct the cell list in the DMBR bank.

→ OBJ2MUO, MUCD, MUMA and DIMU (from the muon library), to get the decisions on muon matching of the "old" phase2, in order to fill the words **QualS1** and **QualS2** of the 501 blocklet.

→ MUEVENTOTH (from muon library) and OTHCLUSTER to flag the out-of-time ECAL neutral clusters using trigger scintillator and muon chamber information.

→ DstFlagMuHits to do the flagging of muon hits associated to identified muons.

→ DstFormTrdHL to form sorted lists of TRD "hits" fitted to DCH-TRD matched tracks: the number of tracks using each hit is also recorded. This information is used to flag "shared hits" in the MAS 201 blocklet *(q.v.)*.

Of the routines to create the individual banks, **CRRVXS**, **CRLEPS**, **CRSVXS**, **CRSTKS** and **CRSCAS** all work in the same manner. Taking **CRRVXS** as an example, it calls the following:

→ RVXSBKD to work out the bank descriptor.

→ MZIOBK/MZLIFT to raise the bank.

→ FILLRVXS to fill the bank.

The naming convention **xxxxBKD**, **FILLxxxx** is preserved for the other banks.

CRDST does the booking and filling of the (trivial) DST bank in one routine. **CREVS** books and raises the EVS bank, but filling is accomplished by **FILLEVS** after all of the other banks have been raised, since information in those banks is used.

The creation of the chain of MAS banks is more involved than the creation of the other banks. It is handled by the routine **CRMAS**, which is essentially a loop over the VTX/TRK structure, checking the MO bank which hangs from every TRK bank for output information from the matching engine.

For each match object, a status word is constructed which flags which subdetectors have contributed information to the match. One subdetector is deemed to be the "seed" subdetector; the one which is driving the matching (this would be the drift chambers for a charged reconstructed track, the calorimeter for an isolated calorimeter cluster, and so on). In addition, the presence of information relating to the alternative clustering of Gary Feldman, of extrapolator information, or of Padova bremsstrahlung information is tested for, and the status word appropriately adjusted. Once the status word is complete, the routine **BKMAS** is called which performs the task of construction of the bank descriptor (calls **MASBKD**), raising of the MAS bank (calls **MZIOBK** and **MZLIFT**) and its filling (calls **FILLMAS**).

When filling a MAS bank, which is made up of a collection of blocklets, the **FILLMAS** routine calls specialised routines for each particular type of MAS blocklet. These specialised routines are named using a logical naming scheme **FILLxxxyyy**, where **xxx** refers to the subdetector providing the information, and **yyy** refers to the seed subdetector. For example, the MAS blocklet of type 401 is filled by the routine **FILLCALDCH**, which may be found in the file **src/fillcaldch.F**.

Auxilliary routines used by the routines mentioned above will be described in section 6.2.3 below. Two additional FORTRAN routines which are used to set the print and debug levels of the DST package are

- **dstSETPRFLAG** for the print level.
- **dstSETDBFLAG** for the debug level.

It is these routines which are called by RECON to set the appropriate print levels based on the values given for the datacard **DSTF**.

6.2.2 Printing of the DST structure

Printing of the DST structure is accomplished by the FORTRAN routine **PrintDST**. This in turn calls specialised routines for each bank, as follows:

→ PrintEVS to print the contents of the EVS bank.

→ PrintRVXS to print the contents of the RVXS bank.

→ PrintMAS to print the chain of MAS banks.

→ PrintLEPS to print the contents of the LEPS bank.

→ PrintSVXS to print the contents of the SVXS bank.

→ PrintSTKS to print the contents of the STKS bank.

→ PrintSCAS to print the contents of the SCAS bank.

As in the case of the **FILLMAS** routine, the **PrintMAS** routine calls specialised routines for each particular type of MAS blocklet. These specialised routines are named using a logical naming scheme **Printxxxyyy**, where **xxx** refers to the subdetector providing the information, and **yyy** refers to the seed subdetector.

6.2.3 Auxilliary routines for filling of the DST structure

A number of auxilliary routines are invoked by the routines already described when creating the DST structure. A brief description of these will now be given.

Bank navigation in FORTRAN is aided through a set of integer functions with the generic name **xxxxREF()**, where **xxxx** is the name of a bank in the DST structure (except for the MAS bank). These functions find the ZEBRA link to the relevant bank and check that the character name of the bank is correct. A few similar functions to find links to certain other banks in the NOMAD ZEBRA structure are also available. These are **DBHCREF()**, **DEHCREF()**, **DOHCREF()**, **LEPTREF()** and **LUJTREF()**. One further function, **DstLinkFromRef(ADR)**, converts a pointer to a bank in the DST structure to its ZEBRA link.

The following FORTRAN routines are used when filling the EVS bank.

- **SUBROUTINE DstGetDensity(NDCVETO, NDCHIT, NDENSITY, NTRDHIT)** Extract information relating to density cuts from the BDCH bank.
- **REAL*8 FUNCTION EnergyCalCells()** Return the total energy deposited in calorimeter cells. The information is taken from the DACA bank.
- **REAL*8 FUNCTION EnergyHcaSlabs()** Return the total energy deposited in the hadron calorimeter. The information is taken from the DAHC bank.
- **INTEGER FUNCTION NumCalCells()** Return the number of calorimeter cells in the event with energy above threshold. The DACA bank is accessed.
- **INTEGER FUNCTION NumDchHitsLeft()** Return the number of drift chamber hits which have not been used in the track building. The DDCH bank is accessed.
- **INTEGER FUNCTION NumHang()** Return the number of hangers (tracks with beginning vertex type 9) in the event. The VTX structure is looped over.
- **INTEGER FUNCTION NumHcaSlabs()** Return the number of hadron calorimeter slabs in the event which have been hit. The DAHC bank is accessed.
- **INTEGER FUNCTION NumMuMatch()** Return the number of “good” muons in the event. The MAS banks are looped over, and for those containing a 501 blocklet, the **ProbMu** word in the corresponding 101 blocklet is checked. If it is set to 1.0, the track is counted as a “good” muon.
- **INTEGER FUNCTION NumPrimChgTracks()** Return the number of charged tracks attached to the primary vertex. The MAS banks are looped over.
- **INTEGER FUNCTION NumPrimNeuTracks()** Return the number of neutral tracks attached to the primary vertex. The MAS banks are looped over.
- **INTEGER FUNCTION NumTrigHits(IPLANE)** Return the number of in-time trigger scintillator hits in the event. The DASC bank is accessed.

- **INTEGER FUNCTION NumUnassCalClus()** Return the number of unassociated calorimeter clusters in the event. Ths MAS banks are looped over to determine this.
- **INTEGER FUNCTION NumUnassHcaClus()** Return the number of unassociated hadron calorime ter clusters in the event. Ths MAS banks are looped over to determine this.
- **INTEGER FUNCTION NumVERT()** Return the number of vertices in the event which are primary or which are secondary but have more than one emergent track (charged or neutral). The VTX structure is looped over.
- **INTEGER FUNCTION NumVetoHits()** Return the number of in-time veto scintillator hits in the event. The MAS banks are looped over to determine this.

The following auxilliary routines are used in filling the RVXS bank.

- **INTEGER FUNCTION NumAssChgTracks(LVTX)** Return the number of outgoing charged tracks associated to a given vertex. The VTX/TRK structure is accessed.
- **INTEGER FUNCTION NumAssNeuTracks(LVTX)** Return the number of outgoing neutral tracks associated to a given vertex. The VTX/TRK structure is accessed.
- **INTEGER FUNCTION NumRVXS()** Return the number of reconstructed vertices in the event. The VTX chain of banks is accessed for this task. Note that end point vertices (type 10) and various dummy vertices (types 11, 0, or -1) are not counted.

In filling the blocklets in the MAS banks, the **FILLxxxyyy** routines make use of a number of utility routines which extract information from the phase 2 ZEBRA banks. The routines which have a name of the form **DstGet...** generally return a real array of information. The **Num...** routines are generally integer functions. All routines are FORTRAN. The full list is as follows:

- **INTEGER FUNCTION ChkFelBremObj(IDOBJ, IFLAG)** Check whether a specified object is associated to any Gary Feldman bremsstrahlung strip in the event. The ID of the track which initiated the strip is returned if an association is found.
- **INTEGER FUNCTION DstCalTFlag(ISTAT)** Extract calorimeter timing information from the status word of the DOCA bank.
- **SUBROUTINE DstGetBestMu(LDMMU, LDOMU, IDMOFF1, IDMOFF2, IDOOFFX1, IDOO IDOOFFX2, IDOOFFY2)** Auxilliary routine to **DstGetDMMU** to obtain pointers to the relevant TKU and TUP entries in the DMMU and DOMU banks for a given matched muon.
- **SUBROUITNE DstGetDcVeto(LTRK, NDCVHITSB, NDCVHITSF)** Obtain information on hits in the drift chamber veto plane around forward and backward extrapolations of a track.
- **SUBROUTINE DstGetDMBR(LTRK, VECT, LDMBR)** Extract information from the DMBR bank.
- **SUBROUTINE DstGetDMCA(LTRK, VECT)** Extract information from the DMCA/DOCA banks.

- **SUBROUTINE DstGetDMFC(LTRK, VECT)** Extract information from the DMFC/DOFC banks.
- **SUBROUTINE DstGetDMHC(LTRK, VECT)** Extract information from the DMHC/DOHC banks.
- **SUBROUTINE DstGetDMMU(LTRK, VECT)** Extract information from the DMMU/DOMU banks.
- **SUBROUTINE DstGetDMPS(LTRK, VECT)** Extract information from the DMPS/DOPS banks.
- **SUBROUTINE DstGetDMTR(LTRK, VECT)** Extract information from the DMTR/DOTR/DATR banks.
- **SUBROUTINE DstGetDMVE(LTRK, VECT)** Extract information from the DMVE bank.
- **SUBROUTINE DstGetFcaMods(NMOD, FCAMOD)** Extract information on a given FCAL module from the FCCL bank.
- **SUBROUTINE DstGetFelBremObj(ITRAK, NOBJ, NTRK, ESTRIP, FELOBJ)** Extract the list of identifiers of objects forming the Gary Feldman bremsstrahlung strip from the DBHC bank.
- **SUBROUTINE DstGetMuExtrap(LTRK, VECT)** Extract extrapolator information at the muon stations from the TXD banks (not used - see **DstGetTXD** instead).
- **SUBROUTINE DstGetMuTrk(NTKU, MUOTKU)** Extract information on muon standalone tracks (TKUs) from the DOMU bank.
- **SUBROUTINE DstGetMuVeto(LTRK, VECT)** Extract information from the muon veto blocklet in the DMMU bank.
- **SUBROUTINE DstGetTXD(LTRK, IDET, IPLANE, VECT, ISTAT)** Extract extrapolator information from the TXD banks.
- **SUBROUTINE DstMuGap(LTRK, PROBGAP)** Return the probability that a given track has passed through the gap in station 1 of the muon chambers. This routine uses the TXD bank information and the muon library routine **GetProbToHitMuGap**.
- **INTEGER FUNCTION FelClusCharged(ITRAK)** Check whether there is a Gary Feldman calorimeter cluster associated with a given track. The DEHC bank is accessed.
- **SUBROUTINE FelMASNeutrals** Raise MAS banks for each of the unassociated Gary Feldman neutral calorimeter clusters. The DEHC bank is accessed.
- **INTEGER FUNCTION IndexInRVXS(ID)** Return the index (or “row number”) in the RVXS bank of a vertex with a given identifier. Used to fill the 101 and 104 blocklet words **IndxVxsB** and **IndxVxsE**.
- **INTEGER FUNCTION IndexInSTKS(ID)** Return the index (or “row number”) in the STKS bank of a simulated track with a given identifier. Used to fill the 101 and 104 blocklet word **IndxStks**.

- **INTEGER FUNCTION NumFcaMods()** Return the number of front calorimeter modules with energy deposition in them for this event. The FCCL bank is accessed.
- **INTEGER FUNCTION NumFelBremObj(ITRAK)** Return the number of objects which contribute to the Gary Feldman bremsstrahlung strip. The DBHC bank is accessed.
- **INTEGER FUNCTION NumMuTrk()** Return the number of standalone muon tracks (TKU objects) in the present event. The DOMU bank is accessed.
- **INTEGER FUNCTION NumPadBrem()** Return the number of Padova bremsstrahlung objects associated to the present track. The DMBR bank is accessed. Note that the number returned is the number of actual bremsstrahlung objects **plus** the number of objects in the cell list, both of which are stored in the present implementation of the DMBR bank.
- **SUBROUTINE ZTRACK˙CARTESIAN(ST, SET, SP, SEP)** An interface to the extrapolator package routine **track˙cartesian**. It is necessary because the extrapolator works in double precision and the ZEBRA structures contain single precision words. It is used when converting from momenta stored as $(1/p, t_x, t_y)$ to (p_x, p_y, p_z).

The following auxilliary routine is used in filling the LEPS bank.

- **INTEGER FUNCTION NumLEPS()** Return the number of simulated particles LEPTO has produced for the event. This is obtained from the LUJT bank.

The following auxilliary routines are used in filling the SVXS bank.

- **SUBROUTINE MuEndInit** Called by CRSVXS it loops over the full SVTX/STRK structure to find muon end vertices and stores their pointers into a common block (MUENDS).
- **LOGICAL FUNCTION MuEndVtx(LSVTX)** Used by FilterSVTX to check if a vertex is a muon end vertex.
- **INTEGER FUNCTION FilterSVTX()** Flag whether a given simulated vertex should be included in the SVXS bank. The selection is presently rather crude - the vertex type should not be end point (type 10) and the z position of the vertex should not be beyond the front face of the hadron calorimeter. If a type 10 vertex is the end point of a muon track, it is retained. Note that this function uses the SVTX/STRK structure directly, and replaces the C function **SelectSVX** of v7r2.
- **int NumChargedTracks(˙DcVertex vertex)** Return the number of simulated charged tracks produced at a simulated vertex. The C **Vxtrack** list of structures, into which the GENOM output data has been unpacked upon input of an event, is accessed to perform this task. Note that this function is not used in the present version of the DST package. Instead the Fortran function **NumChgdSecSVTX** is used, which accesses the SVTX/STRK structure directly. (in **utilfilldst.c**).
- **int NumNeutralTracks(˙DcVertex vertex)** Return the number of simulated neutral tracks produced at a simulated vertex. The C **Vxtrack** list of structures, into which the GENOM output data has been unpacked upon input of an event, is accessed to perform this task. Note that this function is not used in the present version of the DST package. Instead the Fortran function **NumNeutSecSVTX** is used, which accesses the SVTX/STRK structure directly. (in **utilfilldst.c**)

- **INTEGER FUNCTION NumChgdSecSVTX()** Return the number of simulated charged tracks produced at a simulated vertex. The SVTX/STRK structure is accessed.
- **INTEGER FUNCTION NumNeutSecSVTX()** Return the number of simulated neutral tracks produced at a simulated vertex. The SVTX/STRK structure is accessed.
- **INTEGER FUNCTION NumSVXS()** Return the number of simulated vertices in the event. The selection function **FilterSVTX** is applied to a track before it is counted. The SVTX/STRK structure is accessed to perform this task.

The following auxilliary routines are used in filling the STKS bank.

- **int dstkfcharge3(int kf)** Given a particle number (KF code) as specified in the Particle Data Group scheme (used by LEPTO and JETSET), returns the charge of the particle. Thanks to Bruce Yabsley for providing the original function. (in **dstlepto.c**)
- **INTEGER FUNCTION DstSTRKCharge(ITYPE)** Return the charge of a track given its Monte Carlo type as stored in the STRK bank. This function uses the C function **dstkfcharge3** described above.
- **void DstTrackGetPBeg(˙DcTrack dctrack, double p[3])** Return the 3-momentum at the beginning vertex of a simulated track. Note that this function is not used in the present version of the DST package. (in **utilfilldst.c**)
- **void DstTrackGetPFirstHit(˙DcTrack dctrack, double p[3])** Return the 3-momentum at the first hit on a simulated track. (in **utilfilldst.c**)
- **void DstTrackGetPLastHit(˙DcTrack dctrack, double p[3])** Return the 3-momentum at the last hit on a simulated track. (in **utilfilldst.c**)
- **void DstTrackGetPEnd(˙DcTrack dctrack, double p[3])** Return the 3-momentum at the end vertex of a simulated track. Note that this function is not used in the present version of the DST package. (in **utilfilldst.c**)
- **void DstTrackGetVFirstHit(˙DcTrack dctrack, double p[3])** Return the vertex position of the first hit on a simulated track. (in **utilfilldst.c**)
- **void DstTrackGetVLastHit(˙DcTrack dctrack, double p[3])** Return the vertex position of the last hit on a simulated track. (in **utilfilldst.c**)
- **INTEGER FUNCTION FilterSTRK()** Flag whether a given simulated track should be included in the STKS bank. The selection is presently rather crude - the z position of the beginning and end vertices of the track should not be beyond the preshower, and the track momentum should be greater than 30 MeV. If thetrack is a muon, it is retained regardless of the above criteria. Note that this function uses the SVTX/STRK structure directly, and replaces the C function **SelectSTK** of v7r2.
- **INTEGER FUNCTION IndexInSVXS(ID)** Return the index (or “row number”) in the SVXS bank of a simulated vertex with a given identifier. Used to fill the words **IndxVtxB** and **IndxVtxE**.

- **INTEGER FUNCTION NumSTKS()** Return the number of simulated tracks in the event. The selection function **FilterSTRK** is applied to a track before it is counted. The SVTX/STRK structure is accessed to perform this task.
- **INTEGER FUNCTION STRKPointerFromLink(LSTRK)** Given the ZEBRA link to an STRK bank, return the pointer to the corresponding C structure into which it has been unpacked.

The following auxilliary routine is used in filling the SCAS bank.

- **INTEGER FUNCTION NumSCAS()** Return the number of simulated calorimeter cell hits in the event. This is obtained from the RCAL bank. Note that a physical calorimeter cell can have a number of hits, each of which is counted.

The following “analysis” routines, related to muon veto studies, have been added to the DST library for the present time as they do not have a residing place in other NOMAD libraries (authors L. DiLella, F. Salvatore).

- **SUBROUTINE MUCAT(IDTK,MUVETO)** A routine to identify whether a track is a muon undergoing a catastrophic interaction.
- **SUBROUTINE TwoLineDist(R1,R2,U1,U2,R,DIST)** A routine to calculate the distance between two lines in 3-dimensional flat space.
- **void veto(˙Mas Mas), int firstvar(), int secondvar()** Functions related to formation of a muon veto based on Dst information. (in **veto.c**)

The following routines appear in the DST library but are not currently used in the package. They are listed here for completeness.

- **SUBROUTINE ChkObj(IRET)** A (presently dummied) routine to make selections on which objects in the match table will become MAS banks on the DST. At present no special selections are made.
- **INTEGER FUNCTION FindSVTX(LSVTX)** Find the position in the linear chain of SVTX banks of the vertex with given ZEBRA link.
- **INTEGER FUNCTION GeantCharge(ID)** Return the charge of a particle with a given GEANT particle code.
- **int TrdMatchDebugLevelSet(int level)** Dummy function in order to get around an undefined reference which at one stage showed up in the phase2 library. (in **utilfilldst.c**)
- **SUBROUTINE ZCARTESIAN˙TRACK(SP, SEP, ST, SET)** An interface to the extrapolator package routine **cartesian˙track**. It is provided because the extrapolator works in double precision and the ZEBRA structures contain single precision words. It would be used when converting from momenta stored as (p_x, p_y, p_z) to $(1/p, t_x, t_y)$.

6.2.4 Access routines to the DST structure

The access routines for the DST package have been described in some detail in chapter 4. Here we will restrict ourselves to some technical details of where to find the code.

The automatically generated C functions for bank navigation and single word access to the DST structure may be found in the file **dstgen.c**. The automatically generated access functions for the package of Fergus Wilson [2] can be found in the file **dstaccess.c**. Neither of these files should be edited directly, but rather the awk scripts described below should be used to regenerate them if necessary.

The C access functions which are **not** generated automatically can be found in the file **utilaccessdst.c**. These are meant to be "public domain", as opposed to the access functions which are used in generation of the DST, which are collected together in **utilfilldst.c**. These latter functions are really meant to be internal to the package (i.e. can change without notice), so users of them should do so aware of this distinction.

The high level FORTRAN access routines may be found in the following files:

- **dstgetblock.F** Access to the contents of the EVS, RVXS, LEPS, SVXS, STKS or SCAS banks.
- **dstgetmasblock.F** Access to the contents of individual blocklets (fixed length part) within a MAS bank.
- **dstgetmassubblock.F** Access to the contents of an individual subblocklet within a variable length MAS blocklet.
- **getdstversion.F** A REAL function returning the DST version number in decimal format.

In order to be sure that the library version which is being used to access the DST structure matches the DST structure that has been read in, one further routine is used internally by the code. This is the FORTRAN routine **CheckDSTVersion**, which basically flags whether or not the library version and DST ZEBRA structure are compatible. This routine can of course be found in the file **checkdstversion.F**.

6.3 Histograms

In the DST source code, C code may be found which will histogram all words on the DST. This code has not been included in the dst library, but users are welcome to take this code "as is" and compile the code with their camel job. The code is generated automatically from a given bankdoc. Instructions for doing this may be found in section 6.4.

Entry points to the code are:

- **DstHistoFileOpen(int lun, char *Filename)** will attach a file named **Filename** on logical unti **lun** into which the histograms are placed. This can be called in FORTRAN for example from CMUHIN (e.g. CALL fDstHistoFileOpen(60, 'dsthisto.hbook'))
- **DstHistoBook()** to book all histograms. This can also be called from CMUHIN (e.g. CALL fDstHistoBook()).
- **DstHistoFill()** to fill all histograms. This can be called from CMUPROC (e.g. CALL fDstHistoFill()).
- **DstHistoFileClose()** will output the histograms to the attached file. It could be called from CMUEND (e.g. CALL fDstHistoFileClose()).

There is also in this directory a kumac **dsthisto.kumac** which may help in displaying the histograms with PAW.

The automatic generation of this code from a bankdoc requires the specification of the binning used for the histograms, and this of course depends on the nature of the word being histogrammed. In the file **dsthisto.ranges** will be found a set of bin parameters for every word which appears in the present or past DSTs, along with a specification as to whether the histogram should be displayed with log or linear y axis. This file is used when generating the code, and can be tailored if desired by the user to produce different binning for the histograms (see next section).

6.4 Awk scripts for automatic code generation

Extending the procedure that has been employed in the **dstaccess** package [2], awk scripts have been written which take as input a bankdoc file dst.banks, and automatically generate C structures which map onto the ZEBRA banks, access code to these structures, a a file of parameters which specify in FORTRAN the DST banks, and code to histogram all DST words. By employing this method of code generation it is hoped to reduce the number of errors in the DST library and to substantially ease maintenance.

The following awk scripts are provided in the subdirectory **scripts**

- **dstaccess.awk** Generates dstaccess.h, dstaccess.c and dstaccess.inc
- **dstbanks.awk** Generates dstbanks.h
- **dstgen.awk** Generates dstgen.h, dstgen.c and dstgen.inc
- **dstparams.awk** Generates dstparams.inc
- **dsthisto.awk** Generates dsthisto.h, dsthisto.c, a dstXXXXhisto.c file for each bank XXXX, and dsthisto.kumac

A Bourne shell script, **dstauto.sh** is also provided, which may be used to generate the automatically generated code by running all of the above awk scripts. This file may be found in the **scripts** subdirectory. Help is given if this is run without arguments. It may be used either to generate these files in the current working directory, or else to install them in a standard nominated DST directory tree. Suppose one wishes to install the code in the current directory, based on the library v7r4. The command to issue is

```
dstauto.sh /nomad/src/dst/v7r4
```

This will use the bankdoc files of extension **.bank** in the directory /nomad/src/dst/v7r4/doc, and the histogram binning file /nomad/src/dst/v7r4/histos/dsthisto.ranges. If this latter file is present in the current working directory, it will be used in preference to the corresponding file above. This allows the user to customize the histogram binning parameters and to regenerate the histogram code.

6.5 Making changes to the DST code

The DST package has evolved in order to make changes to the code as straightforward as possible. The main thrust of this effort has been in the direction of generating much of the access code automatically through the use of awk scripts, as was described above. These awk scripts use the ZEBRA bankdoc files (the **.bank** files stored in the **doc** subdirectory) as input. Therefore, to add or subtract words from the DST, or alter the ZEBRA banks in any way, the correct procedure to follow is to edit the appropriate bankdoc file for the bank in question, and then run the shell script **dstauto.sh**, as described above, to regenerate much of the source code.

Clearly the filling code for the new or altered words cannot be done automatically, so the next step is to edit the appropriate **FILL...** routine, be it for a complete bank or for a MAS blocklet, in order to correctly fill the words. Because the **PRINT...** routines contain FORMAT statements which explicitly lay out the bank contents in the output listing, these must be edited also and the FORMAT statement changed. In general, the rest of the routine will not need to be changed since in most cases it is written using the parameters in **dstparams.inc**. It is best to check however.

When the DST format is changed, it should be remembered that the library will need to be recompiled from scratch, since the commons or structures in the include files will have changed. Building of the library will be discussed in the following section.

6.6 Building the DST library

The DST library is built in a completely analogous manner to the majority of the other libraries in the NOMAD software, using a Makefile conforming to the standard template.

Assuming that one has a checked out version of the repository code, or the directory structure for a particular version of the DST package, the procedure to follow is the following.

- Go to the top directory of the package (i.e. the directory which contains the **Makefile**). This will be the **dst** directory if working with a checked out version of the code, or the directory e.g. **dst/v7r4** if working with an export of a particular version.
- Issue the command

  ```
  gmake version
  ```

  which ensures that the file **include/dstversion.inc** contains the current version number of the package.
- Issue the command

  ```
  gmake depend
  ```

  which generates the correct version of the file **Make.depends** for the current directory structure.
- Issue the command

  ```
  gmake
  ```

  to build the library.

If the set of environment variables $USERSRCDIR, $USERLIBDIR and $USERBINDIR were defined in the shell in which the library is being built, then the library will appear in the directory $USERLIBDIR (this is often the case if working with a private development directory and checked out code). If not, then the library will appear in a subdirectory of the directory in which the Makefile resides, the directory having the name of the Unix flavour being used **OSF1**, **Solaris** etc. This is usually the case when working with exported code of a particular library version (as in installing on a machine remote from CERN).

The library will have the name **libdstxxx.a**, where **xxx** is the version number as stored in the file **dstversion.inc** mentioned above.

6.7 Documentation

The LateX files for this document can be found in the **doc** subdirectory, as can the set of files xxx.bank, one for each bank in the structure. These contain the raw material for generating the bank doc. In previous releases these bank descriptors were kept in a single file **dst.banks**.

To generate a bank doc from this file, one must first concatenate the various files, using

```
cat dst.bank evs.bank rvxs.bank mas.bank \
    leps.bank svxs.bank stks.bank scas.bank > dst.banks
```

Then run the CERN Library binary dzeX11, and issue the following commands:

```
createdoc dst.banks dst.rz
drawone   DST  NOMA
exit
```

This will produce a PostScript file dst.ps containing the bankdoc, which can be viewed or printed. Note that it may be necessary, to avoid a warning message, to remove the old dst.rz file, if present in the directory, before following this procedure.

Chapter 7: Release Notes

In this section some release notes for the current version are given.

7.1 v7r4 Release Notes

The present release is designed to be compatible with recon version 7 and its associated libraries. It has particularly been designed to function with the v7r7 version of recon, which was produced following a considerable amount of development on the phase2 library.

7.1.1 v7r3 to v7r4

Two small changes have also been made to the banks:

- The word **Chi2MisM** has been added to the RVXS bank, recording the χ^2 for the hypothesis that a V0 "points" to the primary vertex. The calculation requires both track and vertex error matices, and since the latter are not recorded on the DST, it was not possible to recover this quantity in previous versions of the DST.
- Words **EHit1** to **EHit9** of MAS blocklet 201, which record the energy deposition in the nine TRD planes, now also flag whether these hits are shared with other DCH-TRD matched tracks. For shared hits, the *negative* of the energy deposition is recorded; for unshared hits the true (non-negative) value is recorded.

Previous release notes for earlier versions of the DST library may be found in the corresponding manuals for that version.

7.2 What is missing and why - v7r4

A list of known omissions and the reason for them is now given. Please report any further omissions and anomalies that you may find.

- MAS Blocklet 501.
  - Word 17 (**MuonT0**) - Muon T0. This seems to be identically zero in the phase2 banks at time of release.

7.3 Version 7r4 and Recon

Versions v7r3 and now v7r4 of the DST code is intended to be used to produce DSTs from production runs of recon v7r7 or later and associated libraries. Note in particular that the muon library version v7r5 must be used in conjunction with this version of the DST library. Earlier muon libraries are not compatible. For other libraries, the versions used in the production of DST v7r2 are appropriate.

Although no longer relevant, we note for historical purposes that for production of DSTs from prod4 output, version 1 DST libraries should be used. In particular the v1r1 DST library should be used, this being the latest and last version compatible with prod4 output.

7.4 Future directions and issues

Users' comments on the DST and on where work is needed are of course welcome at any time, although it is expected that any future updates to the DST library at this stage of NOMAD are likely to be rare. In any event, a list of recent, current, pending and suggested tasks with regard to the DST can be consulted at any time by reading the DST Web Page [1].

Bibliography

[1] At the time of writing, the URL for this page is
http://pauli.physics.usyd.edu.au/Public/dst/dst.html

[2] *Nomad DST Utility Routines*. Fergus Wilson. NOMAD-MEMO 96-029
At the time of writing, this document may be found on the cluster in the file $NOMAD˙PS/dstgenv1r2.ps.

[3] *Nomad Reconstruction Software - Nomad DST Package - Version v7r2*. Kevin Varvell. NOMAD-MEMO 97-034.

[4] *Some Extended Tools for Track Breakpoint Analysis (and a speedup of matrix inversion)*. Bob Cousins, NOMAD-MEMO 96-016.

[5] At the time of writing, the URL for this page is
http://nomadinfo.cern.ch/Classified/working˙groups/phase2

[6] *NOMAD TRD Electron Identification: Method and First Results*. T. Fazio, J-P. Mendiburu, P. Nédélec, D. Sillou and S. Valuev, NOMAD-MEMO 95-041.

[7] *NOMAD TRD Identification of Overlapping Tracks*. P. Nédélec, D. Sillou and S. Valuev, NOMAD-MEMO 96-005.

[8] *Parametrization of e and γ initiated showers in the NOMAD lead-glass calorimeter*. R. Petti, NOMAD-MEMO 97-018.

[9] *The Performance of the Hadron Calorimeter*. P. Hurst, NOMAD-MEMO 97-042.

[10] *PYTHIA 5.7 and JETSET 7.4. Physics and Manual*. Torbjörn Sjöstrand. CERN-TH.7112/93, section 5.1.