UCRL-MA-118751

SILO User's Guide

March 7, 1995

Lawrence Livermore National Laboratory

DISCLAIMER

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

C Functions

```
#include <silo.h>
int DBAddFltComponent (Object *object, char
    *compname, double f);
int DBAddIntComponent (Object *object, char
    *compname, int i);
int DBAddOption (DBoptlist *optlist, int
    option, void *value);
int DBAddStrComponent (Object *object, char
    *compname, char *s);
int DBAddVarComponent (DBObject *object, char
    *compname, char *vardata);
DBcompoundarray *DBAllocCompoundarray (void);
DBedgelist *DBAllocEdgelist (void);
DBfacelist *DBAllocFacelist (void);
DBmaterial *DBAllocMaterial (void);
DBmatspecies *DBAllocMatspecies (void);
DBmeshvar *DBAllocMeshvar (void);
DBmultimesh *DBAllocMultimesh (void);
DBmultivar *DBAllocMultivar (void);
DBpointmesh *DBAllocPointmesh (void);
DBquadmesh *DBAllocQuadmesh (void);
DBquadvar *DBAllocQuadvar (void);
DBucdmesh *DBAllocUcdmesh (void);
DBucdvar *DBAllocUcdvar (void);
DBzonelist *DBAllocZonelist (void);
DBfacelist *DBCalcExternalFacelist (int
    znodelist[], int nnodes,
    int origin, int zshapesize[],
    int zshapecnt[], int nzshapes,
    int matlist[], int bnd_method);
int DBClearObject (DBobject *object);
int DBClearOptlist (DBoptlist *optlist);
int DBClose (DBfile *dbfile);
int DBContinue (DBfile *dbfile);
DBfile *DBCreate (char *pathname, int mode, int
    target, char *fileinfo, int filetype);
char *DBErrFunc(void);
int DBErrno(void);
char *DBErrString(void);
int DBFreeCompoundarray (DBcompoundarray *x);
int DBFreeEdgelist (DBedgelist *x);
int DBFreeFacelist (DBfacelist *x);
int DBFreeMaterial (DBmaterial *x);
int DBFreeMatspecies (DBmatspecies *x);
int DBFreeMeshvar (DBmeshvar *x);
int DBFreeMultimesh (DBmultimesh *x);
int DBFreeMultivar (DBmultivar *x);
int DBFreeObject (DBobject *x);
int DBFreeOptlist (DBoptlist *optlist);
int DBFreePointmesh (DBpointmesh *x);
int DBFreeQuadmesh (DBquadmesh *x);
int DBFreeQuadvar (DBquadvar *x);
int DBFreeUcdmesh (DBucdmesh *x);
int DBFreeUcdvar (DBucdvar *x);
int DBFreeZonelist (DBzonelist *x);
void *DBGetAtt (DBfile *dbfile, char *varname,
    char *attname);
void *DBGetComponent (DBfile *dbfile, char
    *objname, char *compname);
DBcompoundarray *DBGetCompoundarray (DBfile
    *dbfile, char *arrayname);
DBcurve *DBGetCurve (DBfile *dbfile, char
    *curvename);
int DBGetDir (DBfile *dbfile, char *dirname);
DBmaterial *DBGetMaterial (DBfile *dbfile, char
    *mat_name);
DBmatspecies *DBGetMatspecies (DBfile *dbfile,
    char *ms_name);
DBmultimesh *DBGetMultimesh (DBfile *dbfile,
    char *meshname);
DBPointmesh *DBGetPointmesh (DBfile *dbfile,
    char *meshname);
DBmeshvar *DBGetPointvar (DBfile *dbfile, char
    *varname);
DBquadmesh *DBGetQuadmesh (DBfile *dbfile, char
    *meshname);
DBquadvar *DBGetQuadvar (DBfile *dbfile, char
    *varname);
DBtoc *DBGetToc (DBfile *dbfile);
DBucdmesh *DBGetUcdmesh (DBfile *dbfile, char
    *meshname);
DBucdvar *DBGetUcdvar (DBfile *dbfile, char
    *varname);
void *DBGetVar (DBfile *dbfile, char *varname);
int DBGetVarByteLength (DBfile *dbfile, char
    *varname);
int DBGetVarLength (DBfile *dbfile, char
    *varname);
int DBInqCompoundarray (DBfile *dbfile, char
    *name, char *elemnames[], int *elemlengths,
    int nelems, int nvalues, int datatype);
int DBInqMeshname (DBfile *dbfile, char
    *varname, char *meshname);
int DBInqMeshtype (DBfile *dbfile, char
    *meshname);
DBobject *DBMakeObject (char *objname, int
    objtype, int maxcomps);
DBoptlist *DBMakeOptlist (int maxopts);
int DBMkDir (DBfile *dbfile, char *dirname);
DBfile *DBOpen (char *name, int type);
int DBPause (DBfile *dbfile);
int DBPutCompoundarray (DBfile *dbfile, char
    *name, char *elemnames[], int *elemlengths,
    int nelems, void *values, int nvalues, int
    datatype, DBoptlist *optlist);
     DBoptlist  *opts ;
int DBPutFacelist (DBfile *dbfile, char *name,
    int nfaces, int ndims, int nodelist[], int
    lnodelist, int origin, int zoneno[], int
    shapesize[], int shapecnt[], int nshapes,
    int types[], int typelist[], int ntypes);
int DBPutMaterial (DBfile *dbfile, char *name,
    char *meshname, int nmat, int matnos[], int
    matlist[], int dims[], int ndims, int
    mix_next[],int mix_mat[], int mix_zone[],
    float mix_vf[], int mixlen, int datatype,
    DBoptlist *optlist);
int DBPutMatspecies (DBfile *dbfile, char
    *name, char *matname, int nspecies_mf, float
    species_mf[], int nmatspec[], int nmat, int
    datatype, int speclist[], int dims[], int
    ndims, int mix_list[], int mixlen, DBoptlist
    *optlist);
int DBPutMultimat (DBfile *dbfile, char *name,
    int nmat, char *matnames[], DBoptlist
    *optlist);
int DBPutMultimesh (DBfile *dbfile, char *name,
    int nmesh, char *meshnames[], int
    meshtypes[], DBoptlist *optlist);
int DBPutMultivar (DBfile *dbfile, char *name,
    int nvar, char *varnames[], int vartypes[],
    DBoptlist *optlist);
int DBPutPointmesh (DBfile *dbfile, char *name,
    int ndims, float *coords[], int nels, int
    datatype, DBoptlist *optlist)
int DBPutPointvar (DBfile *dbfile, char *name,
    char *meshname, int nvars, float *vars[],
    int nels, int datatype, DBoptlist *optlist);
int DBPutPointvar1 (DBfile *dbfile, char *name,
    char *meshname, float var[], int nels, int
    datatype, DBoptlist *optlist);
int DBPutQuadmesh (DBfile *dbfile, char *name,
    char *coordnames[], float *coords[], int
    dims[], int ndims, int datatype, int
    coordtype, DBoptlist *optlist)
```

```
int DBPutQuadvar (DBfile *dbfile, char *name,
    char *meshname,int nvars, char *varnames[],
    float *vars[], int dims[], int ndims, float
    *mixvars[], int mixlen, int datatype, int
    centering, DBoptlist *optlist)
int DBPutQuadvar1 (DBfile *dbfile, char *name,
    char *meshname, float *var, int dims[], int
    ndims, float *mixvar, int mixlen, int
    datatype, int centering, DBoptlist
    *optlist)
int DBPutUcdmesh (DBfile *dbfile, char *name,
    int ndims, char *coordnames[], float
    *coords[], int nnodes, int nzones, char
    *zonel_name, char *facel_name, int
    datatype, DBoptlist *optlist)
int DBPutUcdvar (DBfile *dbfile, char *name,
    char *meshname, int nvars, char *varnames[],
    float *vars[], int nels, float *mixvars[],
    int mixlen, int datatype, int centering,
    DBoptlist *optlist)
int DBPutUcdvar1 (DBfile *dbfile, char *name,
    char *meshname, float *var, int nels, float
    *mixvar, int mixlen, int datatype, int
    centering, DBoptlist *optlist);
int DBPutZonelist (DBfile *dbfile, char
    *name, int nzones, int ndims, int
    nodelist[], int lnodelist, int origin, int
    shapesize[], int shapecnt[], int nshapes);
int DBReadAtt (DBfile *dbfile, char *varname,
    char *attname, void *results);
int DBReadVar (DBfile *dbfile, char *varname,
    void *result);
int DBReadVar1 (DBfile *dbfile, char *varname,
    int offset, void *result);
int DBReadVarSlice (DBfile *dbfile, char
    *varname, int *offset, int *length, int
    *stride, int ndims, void *result);
int DBSetDir (DBfile *dbfile, char *pathname);
void DBShowErrors (int level, void
    (*func)(char*));
int DBWrite (DBfile *dbfile, char *varname,
    void *var, int *dims, int ndims, int
    datatype);
int DBWriteComponent (DBfile *dbfile, DBobject
    *object, char *compname, char *prefix, char
    *datatype, void *var, int nd, long *count);
int DBWriteObject (DBfile *dbfile, DBobject
    *object);
```

Fortran Functions

```
include "silo.inc"
integer function dbaddcopt (optlist_id, option,
    cvalue, lcvalue)
integer function dbadddopt (optlist_id, option,
    dvalue)
integer function dbaddiopt (optlist_id, option,
    ivalue)
integer function dbaddropt (optlist_id, option,
    rvalue)
integer function dbcalcfl (znodelist, nnodes,
    origin, zshapesize, zshapecnt, nzshapes,
    matlist, bnd_method, id)
integer function dbclose (dbid)
integer function dbcreate (pathname, lpathname,
    mode, target, fileinfo, lfileinfo,
    filetype, dbid)
integer function dbfgetca (dbid, name, lname,
    values, nvalues)
integer function dbfreeoptlist (optlist_id)
integer function dbgetca (dbid, name, lname,
    enames, lenames, elengths, nelems, values,
    nvalues, datatype)
integer function dbinqca (dbid, name, lname,
    tlenames, nelems, nvalues, datatype)
integer function dbinqlen (dbid, varname,
    lvarname, len)
integer function dbmkdir (dbid, dirname,
    ldirname, id)
integer function dbmkoptlist (maxopts,
    optlist_id)
integer function dbopen (name, lname, type,
    dbid)
integer function dbputca (dbid, name, lname,
    enames, lenames, elengths, nelems, values,
    nvalues, datatype, optlist_id, id)
integer function dbputfl (dbid, name, lname,
    nfaces, ndims, nodelist,lnodelist, origin,
    zoneno, shapesize, shapecnt, nshapes,
    types, typelist, ntypes, idfl)
integer function dbputmat (dbid, name, lname,
    meshname, lmeshname, nmat, matnos, matlist,
    dims, ndims, mix_next, mix_mat, mix_zone,
    mix_vf, mixlen, datatype, optlist_id, id)
integer function dbputmmat (dbid, name, lname,
    nmat, matnames, lmatnames, optlist_id, id)
integer function dbputmmesh (dbid, name, lname,
    nmesh, meshnames, lmeshnames, meshtypes,
    optlist_id, id)
integer function dbputmsp (dbid, name, lname,
    matname, lmatname, nmat, nmatspec,
    speclist, dims, ndims, nspecies_mf,
    species_mf, mix_speclist, mixlen,
    datatype, optlist_id, id)
integer function dbputmvar (dbid, name, lname,
    mmeshname, lmmeshname, optlist_id, id)
integer function dbputpm (dbid, name, lname,
    ndims, x, y, z, nels, datatype, optlist_id,
    id)
integer function dbputpv1 (dbid, name, lname,
    meshname, lmeshname, var, nels, datatype,
    optlist_id, id)
integer function dbputqm (dbid, name, lname,
    xname, lxname, yname, lyname, zname, lzname,
    x, y, z, dims, ndims, datatype, coordtype,
    optlist_id, id)
integer function dbputqv1 (dbid, name, lname,
    meshname, lmeshname, var, dims, ndims,
    mixvar, mixlen, datatype, centering,
    optlist_id, id)
integer function dbputum (dbid, name, lname,
    ndims, x, y, z, xname, lxname, yname,
    lyname, zname, lzname, datatype, nnodes,
    nzones, zlname, lzlname, flname, lflname,
    optlist_id, id)
integer function dbputuv1 (dbid, name, lname,
    meshname, lmeshname, var, nels, mixvar,
    mixlen, datatype, centering, id)
integer function dbputzl (dbid, name, lname,
    nzones, ndims, nodelist, lnodelist, origin,
    shapesize, shapecnt, nshapes, idzl)
integer function dbrdvar (dbid, varname,
    lvarname, result)
integer function dbrdvarslice (dbid, varname,
    lvarname, offset, length, stride, ndims,
    result)
integer function dbsetdir (dbid, pathname,
    lpathname)
integer function dbshowerrors (level)
integer function dbwrite (dbid, varname,
    lvarname, var, dims, ndims, datatype)
```

Table of Contents

List of Figures

List of Tables

Preface

This manual replaces the former *SILO/SLIDE User's Guide*.

The most significant change has been to enhance the higher level interface functions, formerly known as the SLIDE functions. *We recommend that applications programming interfaces (APIs) be developed with these higher level functions*, *which are called the SILO functions in this edition*.

The former SILO functions have been designated as "netCDF functions" in this edition because they actually are either netCDF[1] functions or extensions of them.

Appendix A contains the constant definitions and data structures for the higher-level, user-application, SILO functions.

The discontinued quadrilateral and UCD functions, as well as the PDBLIB enhancements and the netCDF functions, that appeared in the appendices to the prior SILO/SLIDE User's Manual have been omitted from this version.

1. *netCDF User's Guide*, Russell K. Rew, Unidata Program Center, UCAR, Boulder, CO, or via anonymous FTP at `unidata.ucar.edu.`

Chapter 1 Introduction to SILO

1.1 Overview

SILO is a library which implements an application programing interface (API) designed for reading and writing scientific data. It is a high-level, portable interface that was developed at Lawrence Livermore National Laboratory to address difficult database issues, such as different, incompatible file formats and libraries, most of which used non-standard features of the Cray compilers. In addition, none of the previous libraries had portable binary file formats.

SILO takes advantage of features in net-CDF (Network Common Data Form)[1] and PDB (Portable Data Base), a binary database file format developed at LLNL by Stewart Brown, to build a powerful data access mechanism and to provide a higher level view of the data. It assigns meaning to different types of objects and supports a hierarchical directory structure. Entities managed by the SILO library include not just arrays, but also meshes, mesh variables, material data, and curves. The SILO interface allows the development of generic tools. A general purpose datafile and a set of general mesh descriptions can be defined and used rather than writing tools for each specific datafile and mesh. For example, MeshTV is a general purpose analysis tool which operates on the types of meshes supported by SILO (quadrilateral, UCD, and point).

SILO is used as our standard, portable programming interface to produce simulation code restart files, link files, and files for the graphic output programs that help to visualize simulation results.

---

1. The Unidata Program Center is managed by the University Corporation for Atmospheric Research and sponsored by the National Science Foundation. The current netCDF software can be obtained from anonymous FTP at `unidata.ucar.edu`.

1.2 SILO Architecture

SILO supports four input ports and one output port.

In the input model, SILO uses the netCDF and PDBext (PDBLib extensions) interfaces to read objects written by those interfaces (Fig. 1-1). SILO can also read objects in Taurus formatted files and objects in SDX-formatted data.

In the output model, SILO uses the PDBext interface to write objects to a SILO file.

1.2.1 Reading SILO Files

A subset of the SILO functions contains application-level routines to be used for reading mesh and mesh-related data from a variety of sources, including SILO files, Panacea files, and UNIX sockets. These functions return compound C data structures which represent data in a general way.

1.2.2 Writing SILO files

A subset of the SILO functions contains application-level routines to be used for writing mesh and mesh-related data into SILO files.

In the C interface, the application provides a compound C data structure representing the data. In the Fortran interface, the data is passed via individual arguments.

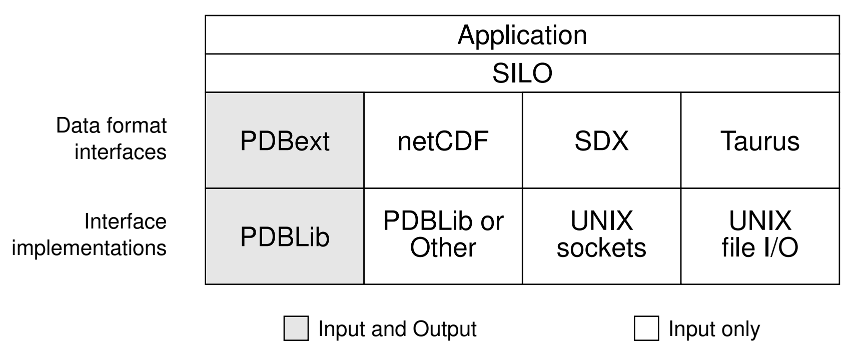


Figure 1-1 Model of SILO Architecture.

1.3 Terminology

Here is a short summary of some of the terms used throughout the SILO interface and documentation. These terms are common to most computer simulation environments.

Block
: This is the fundamental building block of a computational mesh. It defines the nodal coordinates of one contiguous section of a mesh (also known as a mesh-block).

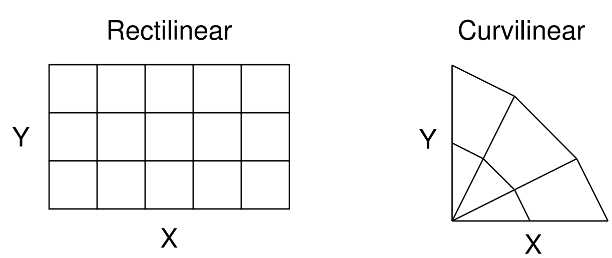
Mesh	A computational mesh, composed of one or more mesh-blocks. A mesh can be composed of mesh-blocks of different types (quad, UCD) as well as of different shapes.
Variable	Data which are associated in some way with a computational mesh. Variables usually represent values of some physics quantity (e.g., pressure). Values are usually located either at the mesh nodes or at zone centers.
Material	A physical material being modeled in a computer simulation.
Node	A mathematical point. The fundamental building-block of a mesh or zone.
Zone	An area or volume of which meshes are comprised. Zones are polygons or polyhedra with nodes as vertices (see "UCD 2-D and 3-D Cell Shapes" on page 1-5.)

1.4 Computational Meshes Supported by SILO

SILO supports three classes, or types, of meshes — quadrilateral, unstructured, and point.

1.4.3 Quadrilateral-Based Meshes and Related Data

A quadrilateral mesh is one which contains four nodes per zone in 2-D and eight nodes per zone (four nodes per zone face) in 3-D. Quad meshes can be either regular, rectilinear, or curvilinear, but they must be logically rectangular (Fig. 1-2).




```
X = {0.0,1.0,2.0,3.0,
     4.0,5.0}
Y = {0.0,1.0,2.0,3.0}
```

```
X = {0.0,1.0,2.0,
     0.0,0.8,1.6,
     0.0,0.4,0.8,
     0.0,0.0,0.0}

Y = {0.0,0.0,0.0,
     0.0,0.4,0.8,
     0.0,0.8,1.6,
     0.0,1.0,2.0}
```

Figure 1-2 Examples of quadrilateral meshes.

A quadrilateral mesh can have “phoney” zones—a layer of zones adjacent to one or more of the “real” mesh boundaries (Fig. 1-3). One use of this feature is to simulate boundary conditions such as a pressure profile.

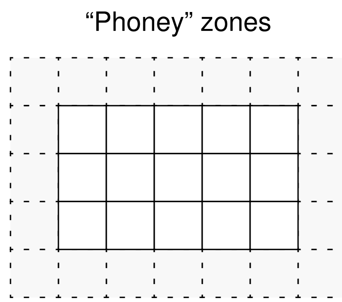


Figure 1-3 Phoney zones around a collinear quadrilateral mesh.

1.4.4 UCD-Based Meshes and Related Data

An unstructured (UCD) mesh is a very general mesh representation; it is composed of an arbitrary list of zones of arbitrary sizes and shapes. Most meshes, including quadrilateral ones, can be represented as an unstructured mesh (Fig. 1-4). Because of their generality, however, unstructured meshes require more storage space and more complex algorithms.

In UCD meshes, the basic concept of zones (cells) still applies, but there is no longer an implied connectivity between a zone and its neighbor, as with the quadrilateral mesh. In other words, given a 2-D quadrilateral mesh zone accessed by *(i, j)*, one knows that this zone’s neighbors are *(i-1,j), (i+1,j), (i, j-1),* and so on. This is not the case with a UCD mesh.

In a UCD mesh, a structure called a zonelist is used to define the nodes which make up each zone. A UCD mesh need not be composed of zones of just one shape (Fig. 1-5). Part of the zonelist structure describes the shapes of the zones in the mesh and a count of how many of each zone shape occurs in the mesh. The facelist structure is analogous to the zonelist structure, but defines the nodes which make up each zone *face*.

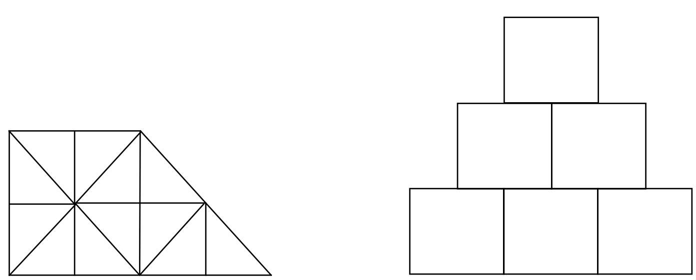


Figure 1-4 Sample 2-D UCD Meshes

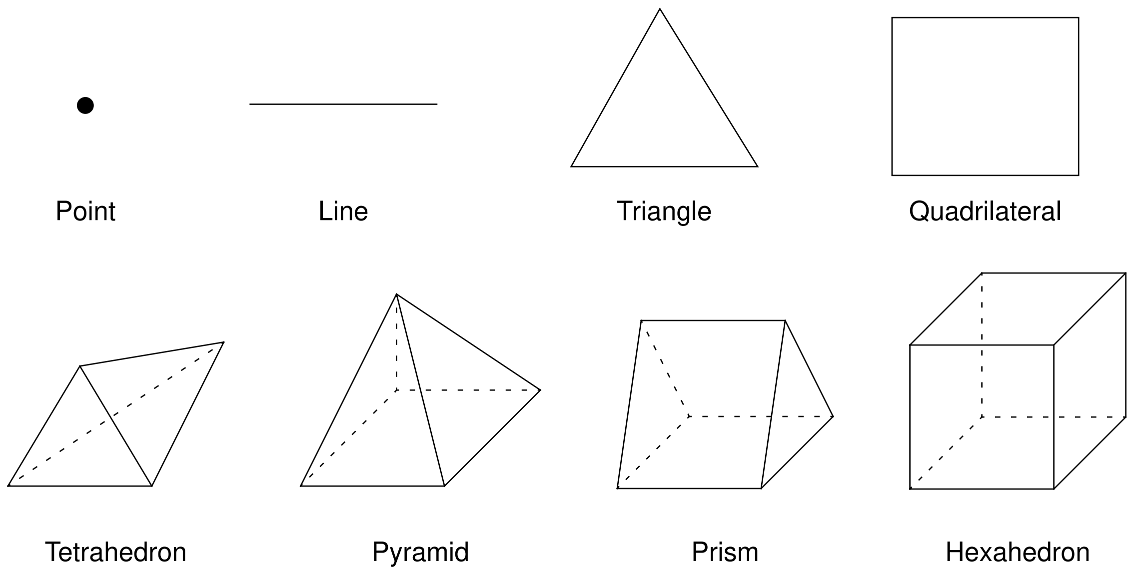


Figure 1-5 UCD 2-D and 3-D Cell Shapes

1.4.5 Point Meshes and Related Data

A point mesh consists of a set of locations, or points, in space. This type of mesh is well suited for representing random scalar data, such as tracer particles.

1.5 SILO Objects

Objects are a grouping mechanism for maintaining related variables, dimensions, and other data. The SILO library understands and operates on specific types of objects including the previously described computational meshes and related data. The user is also able to define arbitrary objects for storage of data if the standard SILO objects are not sufficient.

The objects are generalized representations for data commonly found in physics simulations. These objects include:

Quadmesh — A quadrilateral mesh. At a minimum, this must include the dimension and coordinate data, but typically also includes the mesh's coordinate system, labelling and unit information, minimum and maximum extents, and valid index ranges.

Quadvar — A variable associated with a quadrilateral mesh. At a minimum, this must include the variable's data, centering information (node-centered vs. zone centered), and the name of the quad mesh with which this variable is associated. Additional information, such as time, cycle, units, label, and index ranges can also be included.

Ucdmesh — An unstructured mesh[1]. At a minimum, this must include the dimension, connectivity, and coordinate data, but typically also includes the

	mesh's coordinate system, labelling and unit information, minimum and maximum extents, and a list of face indices.
Ucdvar	A variable associated with a UCD mesh. This at a minimum must include the variable's data, centering information (node-centered vs. zone-centered), and the name of the UCD mesh with which this variable is associated. Additional information, such as time, cycle, units, and label can also be included.
Pointmesh	A point mesh. At a minimum, this must include dimension and coordinate data.
Multimat	A set of materials. This object contains the names of the materials in the set.
Multimesh	A set of meshes. This object contains the names of and types of the meshes in the set.
Multivar	Mesh variable data associated with a multimesh.
Material	Material information. This includes the number of materials present, a list of valid material identifiers, and a zonal-length array which contains the material identifiers for each zone.
Material species	Extra material information. A material species is a type of a material. They are used when a given material (i.e. air) may be made up of other materials (i.e. oxygen, nitrogen) in differing amounts.
Zonelist	Zone-oriented connectivity information for a UCD mesh. This object contains a sequential list of nodes which identifies the zones in the mesh, and arrays which describe the shape(s) of the zones in the mesh.
Facelist	Face-oriented connectivity information for a UCD mesh. This object contains a sequential list of nodes which identifies the faces in the mesh, and arrays which describe the shape(s) of the faces in the mesh. It may optionally include arrays which provide type information for each face.
Curve	X versus Y data. This object must contain at least the domain and range values, along with the number of points in the curve. In addition, a title, variable names, labels, and units may be provided.
Variable	Array data. This object contains, in addition to the data, the dimensions and data type of the array. This object is not required to be associated with a mesh.

1. Unstructured cell data (UCD) is a term commonly used to denote an arbitrarily connected mesh. Such a mesh is composed of vectors of coordinate values along with an index array which identifies the nodes associated with each zone and/or face. Zones may contain any number of nodes for 2-D meshes, and either four, five, six, or eight nodes for 3-D meshes.

1.6 SILO Directories

SILO supports directories as well as objects. Directories allow the user to structure a database into a hierarchy that is analogous to a UNIX™ file system (Fig. 1-6).

In a SILO file, a directory called RootDir represents the top of the hierarchy within a database. Although it is possible for RootDir to be the only directory in a SILO file, in general there are other directories branching from it which in turn may have directories branching from them. There is no limit to the number of branches or levels in the resulting tree structure.

From a given directory one sees only the directories and objects directly below it in the hierarchy. In other words, each SILO directory contains a virtual SILO file. SILO provides a path selection facility to move up and down the hierarchy of directories. In the figure below, if the current position in the hierarchy is "Dir 1", then the only items returned by a "list-contents" call would be "Var 1" and "Dir 4". Note that this concept is analogous to a hierarchical file system: when one issues a "list files" command, only the files in the current directory are listed. By changing directories, one can "see" the contents of other parts of the file system.

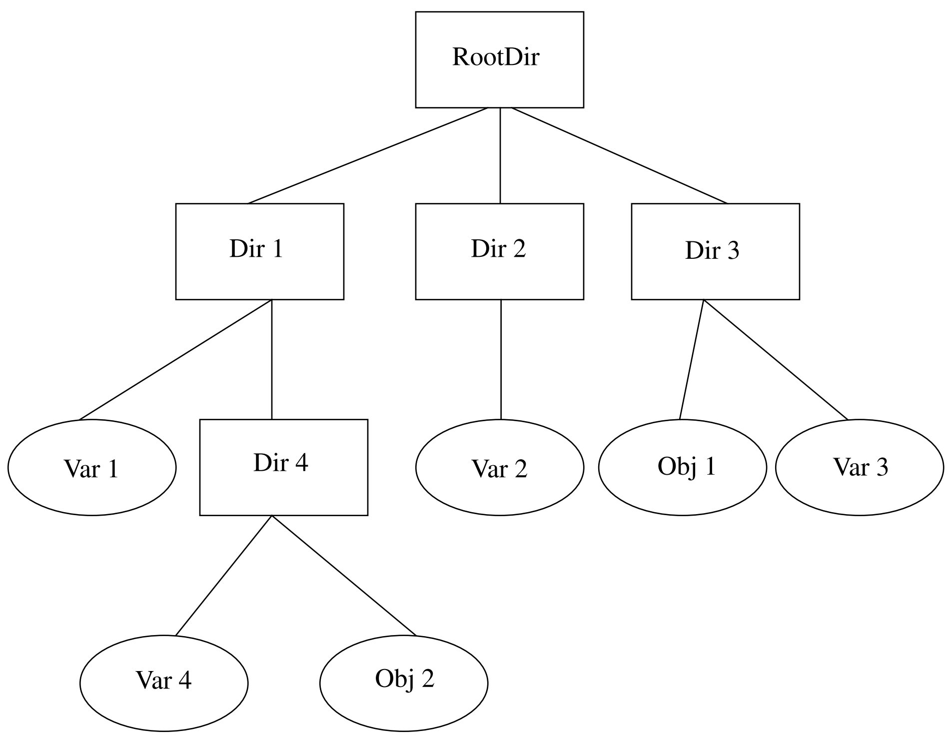


Figure 1-6 Sample hierarchy within a SILO file.

Chapter 2

C Functions

2.1. C Interface Overview

This chapter contains the C interface function summaries for the SILO input and output packages. The interface supports both quadrilateral and UCD-based data. The functions are presented in alphabetical order.

2.1.1. Error Handling

SILO has an error-reporting function DBShowErrors() that allows the programmer to tailor the reporting of errors. This function takes two arguments. The first argument is the error level, which is one of the following values:

Error level value	Error action
DB_ALL	Show all errors, beginning with the (possibly internal) routine that first detected the error and continuing up the call stack to the application.
DB_ABORT	Same as DB_ALL except abort() is called after the error message is printed.
DB_TOP	(Default) Only the top-level SILO functions issue error messages.
DB_NONE	The library does not handle error messages. The application is responsible for checking the return values of the SILO functions and handling the error.

The second argument is a function pointer to an error handing function. This function will be passed a string that is part of the error message (similar to the perror() function). If the function pointer is NULL, then the library will issue error messages to the standard error stream.

The error text and erring function name can be obtained by calling DBErrString() or DBErrFunc(). The internal error number can also be obtained by calling DBErrno().

2.1.2. Optional Arguments

The functions described below may have optional arguments. By optional, it is meant that a dummy value can be supplied instead of an actual value. An argument to a C function which the user does not want to provide, and which is documented as being optional, should be replaced with a NULL (as defined in the file `silo.h`).

2.1.3. Using the SILO Option Parameter

Many of the functions take as one of their arguments a list of option-name/option-value pairs. In this way additional information can be passed to a function without having to change the function's interface. The following sequence of function declarations outlines the procedure for creating and populating such a list:

```
DBoptlist *DBMakeOptlist (int maxopts) /* Create a list with
                                          maximum list length */

int DBAddOption (          /* Add an option to the list: */
           DBoptlist *optlist,   /* the list, */
           int option_id,        /* the option, */
           void *option_value    /* the option's value */
           )
```

2.1.4. C Calling Sequence

The functions in the SILO output package should be called in a particular order.

2.1.4.1. Write Sequence

Start by creating a SILO file, with DBCreate(), create any necessary directories, then call the remaining routines as needed for writing out the mesh, material data, and any physics variables associated with the mesh.

Schematically, your program should look something like this:

```
DBCreate

DBMkdir
DBSetDir
     DBPutQuadmesh
     DBPutQuadvar1
     DBPutQuadvar1
      . . .
DBSetDir

DBMkdir
DBSetDir
```

```
                    DBPutZonelist
                    DBPutFacelist
                    DBPutUcdmesh
                    DBPutMaterial
                    DBPutUcdvar1
                     . . .
               DBSetDir
               DBClose
```

2.1.4.2. Example of C Calling Sequence for writing

The following C code is an example of the creation of a SILO file with just one directory (the root):

```
#include <silo.h>
#include <string.h>

int main()
{
    DBfile         *file = NULL;      /* The SILO file pointer */
    char           *coordnames[2];    /* Names of the coordinates */
    float           nodex[4];         /* The coordinate arrays */
    float           nodey[4];
    float          *coordinates[2];   /* The array of coordinate
                                         arrays */
    int             dimensions[2];    /* The number of nodes in
                                        each dimension */

    /* Create the SILO file */
    file = DBCreate("sample.silo", DB_CLOBBER, DB_LOCAL, NULL,
                     DB_PDB);

    /* Name the coordinate axes 'X' and 'Y' */
    coordnames[0] = strdup("X");
    coordnames[1] = strdup("Y");

    /* Give the x coordinates of the mesh */
    nodex[0] = -1.1;
    nodex[1] = -0.1;
    nodex[2] =  1.3;
    nodex[3] =  1.7;

    /* Give the y coordinates of the mesh */
    nodey[0] = -2.4;
    nodey[1] = -1.2;
    nodey[2] =  0.4;
    nodey[3] =  0.8;

    /* How many nodes in each direction? */
    dimensions[0] = 4;
    dimensions[1] = 4;
```

```
    /* Assign coordinates to coordinates array */
    coordinates[0] = nodex;
    coordinates[1] = nodey;

    /* Write out the mesh to the file */
    DBPutQuadmesh(file, "mesh1", coordnames, coordinates,
                  dimensions, 2, DB_FLOAT, DB_COLLINEAR, NULL);

    /* Close the SILO file */
    DBClose(file);

    return (0);
}
```

2.1.4.3. Read Sequence

Start by opening the SILO file with DBOpen(), then change to the required directory, and then read the mesh, material, and variables. Schematically, your program should look something like this:

```
DBOpen

DBSetDir
      DBGetQuadmesh
      DBGetQuadvar1
      DBGetQuadvar1
       . . .

DBSetDir
      DBGetUcdmesh
      DBGetUcdvar1
      DBGetMaterial
       . . .

DBClose
```

2.1.4.4. Example of C Calling Sequence for reading

The following C code is an example of reading the SILO file in the previous example):

```
#include <silo.h>

int main()
{
    DBfile        *file = NULL; /* The SILO file pointer */
    DBquadmesh    *qm = NULL;   /* The Quadmesh pointer */
    int            i, j;        /* Used for indexing for loops */
    int            ndims;       /* The number of dimensions */
    int           *dims;        /* The array of dimensions */

    /* Open up the SILO file */
```

```
file = DBOpen("sample.silo", DB_UNKNOWN);

/* Get the quadmesh */
qm = DBGetQuadmesh(file, "mesh1");

/* Print out the information, if possible */
if (qm != NULL)
{
    ndims = qm->ndims;
    dims = qm->dims;

    printf("Mesh information:\n");
    printf("(See Appendix A in SILO manual for explanation)\n");
    printf("id: %d\n", qm->id);
    printf("block_no: %d\n", qm->block_no);
    printf("name: \"%s\"\n", qm->name);
    printf("cycle: %d\n", qm->cycle);
    printf("time: %lf\n", qm->time);
    printf("coord_sys: %d\n", qm->coord_sys);
    printf("major_order: %d\n", qm->major_order);
    printf("stride[1-%d]:", ndims);
    for (i = 0; i < ndims; i++)
        printf(" %d", qm->stride[i]);
    printf("\ncoordtype: %d\n", qm->coordtype);
    printf("facetype: %d\n", qm->facetype);
    printf("planar: %d\n", qm->planar);
    for (i = 0; i < ndims; i++)
    {
        printf("dims[%d]:", i);
        for (j = 0; j < dims[i]; j++)
            printf(" %lf", qm->coords[i][j]);
        printf("\n");
    }
    printf("datatype: %d\n", qm->datatype);
    printf("min_extents[1-%d]:", ndims);
    for (i = 0; i < ndims; i++)
        printf(" %lf", qm->min_extents[i]);
    printf("\nmax_extents[1-%d]:", ndims);
    for (i = 0; i < ndims; i++)
        printf(" %lf", qm->max_extents[i]);
    printf("\nlabels[1-%d]:", ndims);
    for (i = 0; i < ndims; i++)
        printf(" \"%s\"", qm->labels[i]);
    printf("\nunits[1-%d]:", ndims);
    for (i = 0; i < ndims; i++)
        printf(" \"%s\"", qm->units[i]);
    printf("\nndims: %d\n", qm->ndims);
    printf("nspace: %d\n", qm->nspace);
    printf("nnodes: %d\n", qm->nnodes);
    printf("dims[1-%d]:", ndims);
```

```
        for (i = 0; i < ndims; i++)
            printf(" %d", qm->dims[i]);
        printf("\norigin: %d\n", qm->origin);
        printf("min_index[1-%d]:", ndims);
        for (i = 0; i < ndims; i++)
            printf(" %d", qm->min_index[i]);
        printf("\nmax_index[1-%d]:", ndims);
        for (i = 0; i < ndims; i++)
            printf(" %d", qm->max_index[i]);
        printf("\n\n");
    }
    else
        printf("Unable to open mesh.\n");

    /* Be safe and close the SILO file */
    DBClose(file);

    return (0);
}
```

DBAddFltComponent()—Add a floating point component to an object.

Synopsis:

```
#include <silo.h>
int DBAddFltComponent (DBobject *object, char *compname, double f)
```

Arguments:

`object`	Pointer to an object. This object is created with the DBMakeObject() function.
`compname`	The component name.
`f`	The value of the floating point component.

Returns:

DBAddFltComponent() returns zero on success and -1 on failure.

Description:

The DBAddFltComponent() function adds a component of floating point data to an existing object.

`DBAddIntComponent()`—Add an integer component to an object.

Synopsis:

```
#include <silo.h>
int DBAddIntComponent (DBobject *object, char *compname, int i)
```

Arguments:

`object`	Pointer to an object. This object is created with the DBMakeObject() function.
`compname`	The component name.
`i`	The value of the integer component.

Returns:

DBAddIntComponent() returns zero on success and -1 on failure.

Description:

The DBAddIntComponent() function adds a component of integer data to an existing object.

`DBAddOption()`—Add an option to an option list.

Synopsis:

```
#include <silo.h>
int DBAddOption (DBoptlist *optlist, int option, void *value)
```

Arguments:

`optlist`	Pointer to an option list structure containing option/value pairs. This structure is created with the DBMakeOptlist() function.
`option`	Option definition. One of the predefined values described in the table in the notes section of each command which accepts an option list.
`value`	Pointer to the value associated with the provided option description. The data type is implied by `option`.

Returns:

DBAddOption() returns a zero on success and -1 on failure.

Description:

The DBAddOption () function adds an option/value pair to an option list. Several of the output functions accept option lists to provide information of an ancillary nature.

DBAddStrComponent()—Add a string component to an object.

Synopsis:

```
#include <silo.h>
int DBAddStrComponent (DBobject *object, char *compname, char *s)
```

Arguments:

`object`	Pointer to the object. This object is created with the DBMakeObject() function.
`compname`	The component name.
`s`	The value of the string component. SILO copies the contents of the string.

Returns:

DBAddStrComponent() returns zero on success and -1 on failure.

Description:

The DBAddStrComponent() function adds a component of string data to an existing object.

DBAddVarComponent()—Add a variable component to an object.

Synopsis:

```
#include <silo.h>
int DBAddVarComponent (DBobject *object, char* compname, char
                    *vardata)
```

Arguments:

`object`	Pointer to the object. This object is created with the DBMakeObject() function.
`compname`	Component name.
`vardata`	Name of the variable object associated with the component (see Description).

Returns:

DBAddVarComponent() returns zero on success and -1 on failure.

Description:

The DBAddVarComponent() function adds a component of the variable type to an existing object.

The variable in `vardata` is stored verbatim into the object. No translation or typing is done on the variable as it is added to the object.

DBAlloc…()—Allocate and initialize a SILO structure.

Synopsis:

```
#include <silo.h>
DBcompoundarray  *DBAllocCompoundarray (void)
DBedgelist       *DBAllocEdgelist (void)
DBfacelist       *DBAllocFacelist (void)
DBmaterial       *DBAllocMaterial (void)
DBmatspecies     *DBAllocMatspecies (void)
DBmeshvar        *DBAllocMeshvar (void)
DBmultimesh      *DBAllocMultimesh (void)
DBmultivar       *DBAllocMultivar (void)
DBpointmesh      *DBAllocPointmesh (void)
DBquadmesh       *DBAllocQuadmesh (void)
DBquadvar        *DBAllocQuadvar (void)
DBucdmesh        *DBAllocUcdmesh (void)
DBucdvar         *DBAllocUcdvar (void)
DBzonelist       *DBAllocZonelist (void)
```

Returns:

These allocation functions return a pointer to a newly allocated and initialized structure on success and NULL on failure.

Description:

The allocation functions allocate a new structure of the requested type, and initialize all values to NULL or zero. There are counterpart functions for freeing structures of a given type (see DBFree…(). See “C Data Structures” on page 3 in Appendix A for a description of these data structures.

DBCalcExternalFacelist()—Calculate an external facelist for a UCD mesh.

Synopsis:

```
#include <silo.h>
DBfacelist *DBCalcExternalFacelist (int znodelist[], int nnodes,
                    int origin, int zshapesize[],
                    int zshapecnt[], int nzshapes, int matlist[],
                    int bnd_method)
```

Arguments:

`znodelist`	Array of node indices describing mesh zones.
`nnodes`	Number of nodes in associated mesh.
`origin`	Origin for indices in the nodelist array. Should be zero or one.
`zshapesize`	Array of length `nzshapes` containing the number of nodes used by each zone shape.
`zshapecnt`	Array of length `nzshapes` containing the number of zones having each shape.
`nzshapes`	Number of zone shapes.
`matlist`	Array containing material numbers for each zone (else NULL).
`bnd_method`	Method to use for calculating external faces. See description below.

Returns:

DBCalcExternalFacelist() returns a DBfacelist pointer on success and NULL on failure.

Description:

The DBCalcExternalFacelist() function calculates an external facelist from the zonelist and zone information describing a UCD mesh. The calculation of the external facelist is controlled by the `bnd_method` parameter as defined in the table below:

bnd_method	Meaning
0	Do not use material boundaries when computing external faces. The matlist parameter can be replaced with NULL.
1	In addition to true external faces, include faces on material boundaries between two clean zones. The matlist parameter must be provided.
2	In addition to true external faces, include faces on material boundaries between two clean zones and between one clean and one mixed zone (only the face from the clean zone will be included.) The matlist parameter must be provided.
4	In addition to true external faces, include faces on material boundaries between two clean zones and between one clean and one mixed zone (both faces will be included in this case.) The matlist parameter must be provided.

Notes:

This function currently only works for quad-shaped zones.

`DBClearObject()`—Clear an object.

Synopsis:

```
#include <silo.h>
int DBClearObject (DBobject *object)
```

Arguments:

`object`	Pointer to the object to be cleared. This object is created with the DBMakeObject() function.

Returns:

DBClearObject() returns zero on success and -1 on failure.

Description:

The DBClearObject() function clears an existing object. The number of components associated with the object is set to zero.

`DBClearOptlist()`—Clear an optlist.

Synopsis:

```
#include <silo.h>
int DBClearOptlist (DBoptlist *optlist)
```

Arguments:

`optlist`	Pointer to an option list structure containing option/value pairs. This structure is created with the DBMakeOptlist() function.

Returns:

DBClearOptlist() returns zero on success and -1 on failure.

Description:

The DBClearOptlist() function removes all options from the given option list.

`DBClose()`—Close a SILO database.

Synopsis:

```
#include <silo.h>
int DBClose(DBfile *dbfile)
```

Arguments:

`dbfile`	Database file pointer.

Returns:

DBClose() returns zero on success and -1 on failure.

Description:

The DBClose() function closes a SILO database.

DBContinue()—Continues a simulation

Synopsis:

```
#include <silo.h>
int DBContinue (DBfile *dbfile)
```

Arguments:

`dbfile`	SILO database pointer

Returns:

DBContinue() returns a zero on success and -1 on failure.

Description:

The DBContinue() function resumes a simulation that was paused with a DBPause() function call.

`DBCreate()`—Create a SILO output file.

Synopsis:

```
#include <silo.h>
DBfile *DBCreate (char *pathname, int mode, int target,
                  char *fileinfo, int filetype)
```

Arguments:

`pathname`	Path name of file to create. This can be either an absolute or relative path.
`mode`	Creation mode. One of the predefined SILO modes: DB_CLOBBER or DB_NOCLOBBER.
`target`	Destination file format. One of the predefined types: DB_LOCAL, DB_SUN3, DB_SUN4, DB_SGI, DB_RS6000, or DB_CRAY.
`fileinfo`	Character string containing descriptive information about the file's contents. This information is usually printed by applications when this file is opened. If no such information is needed, send NULL for this argument.
`filetype`	Destination file type. Currently only one type is supported: DB_PDB.

Returns:

DBCreate() returns a DBfile pointer on success and NULL on failure.

Description:

The DBCreate function creates a SILO file and initializes it for writing data.

Notes:

The underlying database library (PDBLib) supports the concept of targeting output files. That is, a Sun IEEE file can be created on the Cray, and vice versa. If creating files on a mainframe or other powerful computer, it is best to target the file for the machine where the file will be processed. Because of the extra time required to do the floating point conversions, however, one may wish to bypass the targeting function by providing DB_LOCAL as the target.

SILO currently creates only one kind of file, a PDB file. This PDB file contains some special structures for handling objects and directory hierarchies.

Note that regardless of what type of file is created, it can still be read on any machine.

DBErrFunc()—Get name of error-generating function

Synopsis:

```
#include <silo.h>
char *DBErrFunc(void)
```

Returns:

DBErrFunc() returns a `char*` containing the name of the function that generated the last error. It cannot fail.

Description:

The DBErrFunc() function is used to find the name of the function that generated the last SILO error. It is implemented as a macro. The returned pointer points into SILO private space and must not be modified or freed.

DBErrno()—Get internal error number.

Synopsis:

```
#include <silo.h>
int DBErrno (void)
```

Returns:

DBErrno() returns the internal error number of the last error. It cannot fail.

Description:

The DBErrno() function is used to find the number of the last SILO error message. It is implemented as a macro. The error numbers are not guaranteed to remain the same between different release versions of SILO.

DBErrString()—Get error message.

Synopsis:

```
#include <silo.h>
char *DBErrString (void)
```

Returns:

DBErrString() returns a `char*` containing the last error message. It cannot fail.

Description:

The DBErrString() function is used to find the last SILO error message. It is implemented as a macro. The returned pointer points into SILO private space and must not be modified or freed.

DBFree…()—Release memory associated with a SILO structure.

Synopsis:

```
#include <silo.h>
int DBFreeCompoundarray (DBcompoundarray *x)
int DBFreeEdgelist (DBedgelist *x)
int DBFreeFacelist (DBfacelist *x)
int DBFreeMaterial (DBmaterial *x)
int DBFreeMatspecies (DBmatspecies *x)
int DBFreeMeshvar (DBmeshvar *x)
int DBFreeMultimesh (DBmultimesh *x)
int DBFreeMultivar (DBmultivar *x)
int DBFreePointmesh (DBpointmesh *x)
int DBFreeQuadmesh (DBquadmesh *x)
int DBFreeQuadvar (DBquadvar *x)
int DBFreeUcdmesh (DBucdmesh *x)
int DBFreeUcdvar (DBucdvar *x)
int DBFreeZonelist (DBzonelist *x)
```

Arguments:

`x`	A pointer to a structure which is to be freed. Its type must correspond to the type in the function name.

Returns:

These free functions return zero on success and -1 on failure.

Description:

The free functions release the given structure as well as all memory pointed to by these structures. This is the preferred method for releasing these structures. There are counterpart functions for allocating structures of a given type (see DBAlloc…).

The functions will not fail if a NULL pointer is passed to them.

`DBFreeObject()`—Free memory associated with an object.

Synopsis:

```
#include <silo.h>
int DBFreeObject (DBobject *object)
```

Arguments:

`object`	Pointer to the object to be freed. This object is created with the DBMakeObject() function.

Returns:

DBFreeObject() returns zero on success and -1 on failure.

Description:

The DBFreeObject function releases the memory associated with the given object. The data associated with the object's components is not released.

DBFreeObject() will not fail if a NULL pointer is passed to it.

DBFreeOptlist()—Free memory associated with an option list.

Synopsis:

```
#include <silo.h>
int DBFreeOptlist (DBoptlist *optlist)
```

Arguments:

`optlist`	Pointer to an option list structure containing option/value pairs. This structure is created with the DBMakeOptlist() function.

Returns:

DBFreeOptlist() returns a zero on success and -1 on failure.

Description:

The DBFreeOptlist() function releases the memory associated with the given option list. The individual option values are not freed.

DBFreeOptlist() will not fail if a NULL pointer is passed to it.

DBGetAtt()—Allocate space for, and return, an attribute value.

Synopsis:

```
#include <silo.h>
void *DBGetAtt(DBfile *dbfile, char *varname, char *attname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable to which the attribute belongs.
`attname`	Name of the attribute.

Returns:

DBGetAtt() returns a pointer to newly allocated space containing the attribute value on success, and NULL on failure.

Description:

The DBGetAtt() function allocates space for an attribute value associated with a variable, reads the attribute value, and returns a pointer to that space. If the attribute or variable does not exist, NULL is returned.

Notes:

See DBReadAtt() for a non-memory allocating version of this function.

DBGetComponent()—Allocate space for, and return, an object component.

Synopsis:

```
#include <silo.h>
void *DBGetComponent(DBfile *dbfile, char *objname,
                     char *compname)
```

Arguments:

`dbfile`	Database file pointer.
`objname`	Object name.
`compname`	Component name.

Returns:

DBGetComponent() returns a pointer to newly allocated space containing the component value on success, and NULL on failure.

Description:

The DBGetComponent() function allocates space for one object component, reads the component, and returns a pointer to that space. If either the object or component does not exist, NULL is returned. It is up to the application to cast the returned pointer to the appropriate type.

`DBGetCompoundarray()`—Read a compound array from a SILO database.

Synopsis:

```
#include <silo.h>
DBcompoundarray *DBGetCompoundarray (DBfile *dbfile, char
                    *arrayname)
```

Arguments:

`dbfile`	Database file pointer.
`arrayname`	Name of the compound array.

Returns:

DBGetCompoundarray() returns a pointer to a DBcompoundarray structure on success and NULL on failure.

Description:

The DBGetCompoundarray() function allocates a DBcompoundarray structure, reads a compound array from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Compound Array Definition” on page 3 in Appendix A for the definition of DBcompoundarray.

DBGetCurve()—Read a curve from a SILO database.

Synopsis:

```
#include <silo.h>
DBcurve *DBGetCurve(DBfile *dbfile, char *curvename)
```

Arguments:

`dbfile`	Database file pointer.
`curvename`	Name of the curve to read.

Returns:

DBCurve() returns a pointer to a DBcurve structure on success and NULL on failure.

Description:

The DBGetCurve() function allocates a DBcurve data structure, reads a curve from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Curve Definition” on page 4 in Appendix A for the definition of DBcurve.

NOT YET IMPLEMENTED.

`DBGetDir()`—Get the Name of the current directory.

Synopsis:

```
#include <silo.h>
int DBGetDir(DBfile *dbfile, char *dirname)
```

Arguments:

`dbfile`	Database file pointer.
`dirname`	Returned current directory name. The caller must allocate space for the returned name. The maximum space used is 256 characters, including the NULL terminator.

Returns:

DBGetDir() returns zero on success and -1 on failure.

Description:

The DBGetDir() function returns the name of the current directory.

`DBGetMaterial()`—Read material data from a SILO database.

Synopsis:

```
#include <silo.h>
DBmaterial *DBGetMaterial(DBfile *dbfile, char *mat_name)
```

Arguments:

`dbfile`	Database file pointer.
`mat_name`	Name of the material variable to read.

Returns:

DBGetMaterial() returns a pointer to a DBmaterial structure on success and NULL on failure.

Description:

The DBGetMaterial() function allocates a DBmaterial data structure, reads material data from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Material Data Definition” on page 4 in Appendix A for the definition of DBmaterial.

DBGetMatspecies()—Read material species data from a SILO database.

Synopsis:

```
#include <silo.h>
DBmatspecies *DBGetMatspecies(DBfile *dbfile, char *ms_name)
```

Arguments:

`dbfile`	Database file pointer.
`ms_name`	Name of the material species data to read.

Returns:

DBGetMatspecies() returns a pointer to a DBmatspecies structure on success and NULL on failure.

Description:

The DBGetMatspecies() function allocates a DBmatspecies data structure, reads material species data from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Material Species Data Definition” on page 4 in Appendix A for the definition of DBmatspecies.

DBGetMultimesh()—Read a multi-block mesh from a SILO database.

Synopsis:

```
#include <silo.h>
DBmultimesh *DBGetMultimesh(DBfile *dbfile, char *meshname)
```

Arguments:

`dbfile`	Database file pointer.
`meshname`	Name of the multi-block mesh.

Returns:

DBGetMultimesh() returns a pointer to a DBmultimesh structure on success and NULL on failure.

Description:

The DBGetMultimesh() function allocates a DBmultimesh data structure, reads a multi-block mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Multi-Block Mesh Definition” on page 6 in Appendix A for the definition of DBmultimesh.

DBGetPointmesh()—Read a point mesh from a SILO database.

Synopsis:

```
#include <silo.h>
DBpointmesh *DBGetPointmesh(DBfile *dbfile, char *meshname)
```

Arguments:

`dbfile`	Database file pointer.
`meshname`	Name of the mesh.

Returns:

DBGetPointmesh() returns a pointer to a DBpointmesh structure on success and NULL on failure.

Description:

The DBGetPointmesh() function allocates a DBpointmesh data structure, reads a point mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Point Mesh Definition” on page 6 in Appendix A for the definition of DBpointmesh.

DBGetPointvar()—Read a point variable from a SILO database.

Synopsis:

```
#include <silo.h>
DBmeshvar *DBGetPointvar(DBfile *dbfile, char *varname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.

Returns:

DBGetPointvar() returns a pointer to a DBmeshvar structure on success and NULL on failure.

Description:

The DBGetPointvar() function allocates a DBmeshvar data structure, reads a variable associated with a point mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Mesh Variable Definition” on page 5 in Appendix A for the definition of DBmeshvar.

`DBGetQuadmesh()`—Read a quadrilateral mesh from a SILO database.

Synopsis:

```
#include <silo.h>
DBquadmesh *DBGetQuadmesh(DBfile *dbfile, char *meshname)
```

Arguments:

`dbfile`	Database file pointer.
`meshname`	Name of the mesh.

Returns:

DBGetQuadmesh() returns a pointer to a DBquadmesh structure on success and NULL on failure.

Description:

The DBGetQuadmesh() function allocates a DBquadmesh data structure, reads a quadrilateral mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Quad Mesh Definition” on page 7 in Appendix A for the definition of DBquadmesh.

DBGetQuadvar()—Read a quadrilateral variable from a SILO database.

Synopsis:

```
#include <silo.h>
DBquadvar *DBGetQuadvar(DBfile *dbfile, char *varname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.

Returns:

DBGetQuadvar() returns a pointer to a DBquadvar structure on success and NULL on failure.

Description:

The DBGetQuadvar() function allocates a DBquadvar data structure, reads a variable associated with a quadrilateral mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “Quad Variable Definition” on page 7 in Appendix A for the definition of DBquadvar.

DBGetToc()—Get the table of contents of a SILO database.

Synopsis:

```
#include <silo.h>
DBtoc *DBGetToc(DBfile *dbfile)
```

Arguments:

`dbfile`	Database file pointer.

Returns:

DBGetToc() returns a pointer to a DBtoc structure on success and NULL on error.

Description:

The DBGetToc() function returns a pointer to a DBtoc structure, which contains the names of the various SILO object contained in the SILO database. The returned pointer points into SILO private space and must not be modified or freed. Also, calls to DBSetDir() will free the DBtoc structure, invalidating the pointer returned previously by DBGetToc().

Notes:

See “Table of Contents Definiton” on page 8 in Appendix A for the definition of DBtoc.

DBGetUcdmesh()—Read a UCD mesh from a SILO database.

Synopsis:

```
#include <silo.h>
DBucdmesh *DBGetUcdmesh(DBfile *dbfile, char *meshname)
```

Arguments:

`dbfile`	Database file pointer.
`meshname`	Name of the mesh.

Returns:

DBGetUcdmesh() returns a pointer to a DBucdmesh structure on success and NULL on failure.

Description:

The DBGetUcdmesh() function allocates a DBucdmesh data structure, reads a UCD mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “UCD Mesh Definition” on page 9 in Appendix A for the definition of DBucdmesh.

DBGetUcdvar()—Read a UCD variable from a SILO database.

Synopsis:

```
#include <silo.h>
DBucdvar *DBGetUcdvar(DBfile *dbfile, char *varname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.

Returns:

DBGetUcdvar() returns a pointer to a DBucdvar structure on success and NULL on failure.

Description:

The DBGetUcdvar() function allocates a DBucdvar data structure, reads a variable associated with a UCD mesh from the SILO database, and returns a pointer to that structure. If an error occurs, NULL is returned.

Notes:

See “UCD Variable Definition” on page 10 in Appendix A for the definition of DBucdvar.

`DBGetVar()`—Allocate space for, and return, a variable.

Synopsis:

```
#include <silo.h>
void *DBGetVar(DBfile *dbfile, char *varname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable

Returns:

DBGetVar() returns a pointer to newly allocated space on success and NULL on failure.

Description:

The DBGetVar() function allocates space for a variable, reads the variable from the SILO database, and returns a pointer to the new space. If an error occurs, NULL is returned. It is up to the application to cast the returned pointer to the correct data type.

Notes:

See DBReadVar() and DBReadVar1() for non-memory allocating versions of this function.

`DBGetVarBytelength()`—Return the byte length of a variable.

Synopsis:

```
#include <silo.h>
int DBGetVarBytelength(DBfile *dbfile, char *varname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Variable name.

Returns:

DBGetVarBytelength() returns the length of the given variable in bytes on success and -1 on failure.

Description:

The DBGetVarBytelength() function returns the length of the requested variable, in bytes. This is useful for determining how much memory to allocate before reading a variable with DBReadVar(). Note that this would not be a concern if one used the DBGetVar() function, which allocates space itself.

`DBGetVarLength()`—Return the number of elements in a variable.

Synopsis:

```
#include <silo.h>
int DBGetVarLength(DBfile *dbfile, char *varname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Variable name.

Returns:

DBGetVarLength() returns the number of elements in the given variable on success and -1 on failure.

Description:

The DBGetVarLength() function returns the length of the requested variable, in number of elements.

DBInqCompoundarray()—Inquire Compound Array attributes.

Synopsis:

```
#include <silo.h>
int DBInqCompoundarray (DBfile *dbfile, char *name, char
                    *elemnames[], int *elemlengths, int nelems,
                    int nvalues, int datatype)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the compound array.
`elemnames`	Returned array of length nelems containing pointers to the names of the array elements.
`elemlengths`	Returned array of length `nelems` containing the lengths of the array elements.
`nelems`	Returned number of array elements.
`nvalues`	Returned number of total values in the compound array.
`datatype`	Datatype of the data values. One of the predefined SILO data types.

Returns:

DBInqCompoundarray() returns zero on success and -1 on failure.

Description:

The DBInqCompoundarray() function returns information about the compound array. It does not return the data values themselves; use DBGetCompoundarray() instead.

`DBInqMeshname()`—Inquire the mesh name associated with a variable.

Synopsis:

```
#include <silo.h>
int DBInqMeshname(DBfile *dbfile, char *varname, char *meshname)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Variable name.
`meshname`	Returned mesh name. The caller must allocate space for the returned name. The maximum space used is 256 characters, including the NULL terminator.

Returns:

DBInqMeshname() returns zero on success and -1 on failure.

Description:

The DBInqMeshname() function returns the name of a mesh associated with a mesh variable. Given the name of a variable to access, one must call this function to find the name of the mesh before calling DBGetQuadmesh() or DBGetUcdmesh().

`DBInqMeshtype()`—Inquire the mesh type of a mesh.

Synopsis:

```
#include <silo.h>
int DBInqMeshtype(DBfile *dbfile, char *meshname)
```

Arguments:

`dbfile`	Database file pointer.
`meshname`	Mesh name.

Returns:

DBInqMeshtype() returns the mesh type on success and -1 on failure.

Description:

The DBInqMeshtype() function returns the type of the given mesh. The value returned is described in the following table:

Mesh Type	Returned Value
Multi-Block	DB_MULTIMESH
UCD	DB_UCDMESH
Pointmesh	DB_POINTMESH
Quad (Collinear)	DB_QUAD_RECT
Quad (Non-Collinear)	DB_QUAD_CURV

`DBMakeObject()`—Allocate an object of the specified length and initialize it.

Synopsis:

```
#include <silo.h>
DBobject *DBMakeObject (char *objname, int objtype, int maxcomps)
```

Arguments:

`objname`	Name of the object.
`objtype`	Type of object. One of the predefined types: DB_QUADMESH, DB_QUAD_RECT, DB_QUAD_CURV, DB_QUADVAR, DB_UCDMESH, DB_UCDVAR, DB_POINTMESH, DB_POINTVAR, DB_MULTIMESH, DB_MULTIVAR, DB_MATERIAL, DB_MATSPECIES, DB_FACELIST, DB_ZONELIST, DB_EDGELIST, DB_CURVE, DB_ARRAY, or DB_USERDEF.
`maxcomps`	Maximum number of components needed for this object.

Returns:

DBMakeObject() returns a pointer to the newly allocated and initialized object on success and NULL on failure.

Description:

The DBMakeObject() function allocates space for an object of `maxcomps` components.

`DBMakeOptlist()`—Allocate an option list.

Synopsis:

```
#include <silo.h>
DBoptlist *DBMakeOptlist (int maxopts)
```

Arguments:

`maxopts`	Maximum number of options needed for this option list.

Returns:

DBMakeOptlist() returns a pointer to an option list on success and NULL on failure.

Description:

The DBMakeOptlist() function allocates memory for an option list and initializes it. Use the function DBAddOption() to populate the option list structure, and DBFreeOptlist() to free it.

`DBMkDir()`—Create a new directory in a SILO file.

Synopsis:

```
#include <silo.h>
int DBMkDir(DBfile *dbfile, char *dirname)
```

Arguments:

`dbfile`	Database file pointer.
`dirname`	Name of the directory to create.

Returns:

DBMkDir() returns zero on success and -1 on failure.

Description:

The DBMkDir function creates a new directory in the SILO file as a child of the current directory (see DBSetDir()). The directory name may be an absolute path name similar to "`/dir/subdir`", or may be a relative path name similar to "`../../dir/subdir`".

`DBOpen()`—Open an existing SILO file.

Synopsis:

```
#include <silo.h>
DBfile *DBOpen(char *name, int type)
```

Arguments:

`name`	Name of the file to open. Can be either an absolute or relative path.
`type`	The type of file to open. One of the predefined types: DB_SDX, DB_PDB, DB_TAURUS, or DB_UNKNOWN.

Returns:

DBOpen() returns a DBfile pointer on success and a NULL on failure.

Description:

The DBOpen() function opens an existing SILO file. If the file `type` is DB_UNKNOWN, SILO will guess at the file type, getting it right most of the time.

`DBPause()`—Pause a simulation

Synopsis:

```
#include <silo.h>
int DBPause (DBfile *dbfile);
```

Arguments:

`dbfile`	SILO database pointer.

Returns:

DBPause() returns zero on success and -1 on failure.

Description:

The DBPause() function pauses the specified simulation until DBContinue() is called.

DBPutCompoundarray()—Write a Compound Array object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutCompoundarray (DBfile *dbfile, char *name, char
                    *elemnames[], int *elemlengths, int nelems,
                    void *values, int nvalues, int datatype,
                    DBoptlist *optlist);
```

Arguments:

`dbfile`	Database file pointer
`name`	Name of the compound array structure.
`elemnames`	Array of length `nelems` containing pointers to the names of the elements.
`elemlengths`	Array of length `nelems` containing the lengths of the elements.
`nelems`	Number of simple array elements.
`values`	Array whose length is determined by `nelems` and `elemlengths` containing the values of the simple array elements.
`nvalues`	Total length of the `values` array.
`datatype`	Data type of the `values` array. One of the predefined SILO types.
`optlist`	Pointer to an option list structure containing additional information to be included in the compound array object written into the SILO file. Use NULL is there are no options.

Returns:

DBPutCompoundarray() returns zero on success and -1 on failure.

Description:

The DBPutCompoundarray() function writes a compound array object into a SILO file. A compound array is an array whose elements are simple arrays. All of the simple arrays have elements of the same data type, and each have a name.

Often, an application will partition a block of memory into named pieces, but write the block to a database as a single entity. Fortran common blocks are used in this way. The compound array object is an abstraction of this partitioned memory block.

DBPutFacelist()—Write a facelist object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutFacelist (DBfile *dbfile, char *name, int nfaces,
                   int ndims, int nodelist[], int lnodelist,
                   int origin, int zoneno[],int shapesize[],
                   int shapecnt[], int nshapes, int types[],
                   int typelist[], int ntypes)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the facelist structure.
`nfaces`	Number of external faces in associated mesh.
`ndims`	Number of spatial dimensions represented by the associated mesh.
`nodelist`	Array of length `lnodelist` containing node indices describing mesh faces.
`lnodelist`	Length of nodelist array.
`origin`	Origin for indices in nodelist array. Either zero or one.
`zoneno`	Array of length `nfaces` containing the zone number from which each face came. Use a NULL for this parameter if zone numbering info is not wanted.
`shapesize`	Array of length `nshapes` containing the number of nodes used by each face shape (for 3-D meshes only).
`shapecnt`	Array of length `nshapes` containing the number of faces having each shape (for 3-D meshes only).
`nshapes`	Number of face shapes (for 3-D meshes only).
`types`	Array of length `nfaces` containing information about each face. This argument is ignored if `ntypes` is zero, or if this parameter is NULL.
`typelist`	Array of length `ntypes` containing the identifiers for each type. This argument is ignored if `ntypes` is zero, or if this parameter is NULL.
`ntypes`	Number of types, or zero if type information was not provided.

Returns:

DBPutFacelist() returns zero on success or -1 on failure.

Description:

The DBPutFacelist() function writes a facelist object into a SILO file. The name given to this object can in turn be used as a parameter to the DBPutUcdmesh() function.

Notes:

See the write-up of DBPutUcdmesh() for a full description of the facelist data structures. *Note that MeshTV expects this structure to contain descriptions of the external faces only.*

DBPutMaterial()—Write a material data object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutMaterial(DBfile *dbfile, char *name, char *meshname,
                  int nmat, int matnos[], int matlist[],
                  int dims[], int ndims, int mix_next[],
                  int mix_mat[], int mix_zone[], float mix_vf[],
                  int mixlen, int datatype, DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the material data object.
`meshname`	Name of the mesh associated with this information.
`nmat`	Number of materials.
`matnos`	Array of length `nmat` containing material numbers.
`matlist`	Array whose dimensions are defined by `dims` and `ndims`. It contains the material numbers for each single-material (non-mixed) zone, and indices into the mixed data arrays for each multi-material (mixed) zone. A negative value indicates a mixed zone, and its absolute value is used as an index into the mixed data arrays.
`dims`	Array of length `ndims` which defines the dimensionality of the `matlist` array.
`ndims`	Number of dimensions in `matlist` array.
`mix_next`	Array of length `mixlen` of indices into the mixed data arrays (one-origin).
`mix_mat`	Array of length `mixlen` of material numbers for the mixed zones.
`mix_zone`	Optional array of length `mixlen` of back pointers to originating zones (one-origin). Even if `mixlen` > 0, this argument is optional.
`mix_vf`	Array of length `mixlen` of volume fractions for the mixed zones.
`mixlen`	Length of mixed data arrays (or zero if no mixed data is present). If `mixlen` > 0, then the “mix_” arguments describing the mixed data arrays must be non-NULL.
`datatype`	Volume fraction data type. One of the predefined SILO data types.
`optlist`	Pointer to an option list structure containing additional information to be included in the material object written into the SILO file. See the table below for the valid options for this function. If no options are to be provided, use NULL for this argument.

Returns:

DBPutMaterial() returns zero on success and -1 on failure.

Description:

The DBPutMaterial() function writes a material data object into the current open SILO file. The minimum required information for a material data object is supplied via the standard arguments to this function. The `optlist` argument must be used for supplying any information not requested through the standard arguments.

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_LABEL	char *	Character string defining the label associated with material data.	NULL
DBOPT_MAJORORDER	int	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0

The model used for storing material data is the most efficient for MeshTV, and works as follows:

One zonal array, `matlist`, contains the material number for a clean zone or an index into the mixed data arrays if the zone is mixed. Mixed zones are marked with negative entries in `matlist`, so you must take `ABS(matlist[i])` to get the actual 1-origin mixed data index. *All indices are 1-origin to allow* `matlist` *to use zero as a material number.*

The mixed data arrays are essentially a linked list of information about the mixed elements within a zone. Each mixed data array is of length `mixlen`. For a given index *i*, the following information is known about the *i*’th element:

`mix_zone[i]` The 1-origin index of the zone which contains this element.

`mix_mat[i]` The material number of this element

`mix_vf[i]` The volume fraction of this element

`mix_next[i]` The 1-origin index of the next material entry for this zone, else 0 if this is the last entry.

.

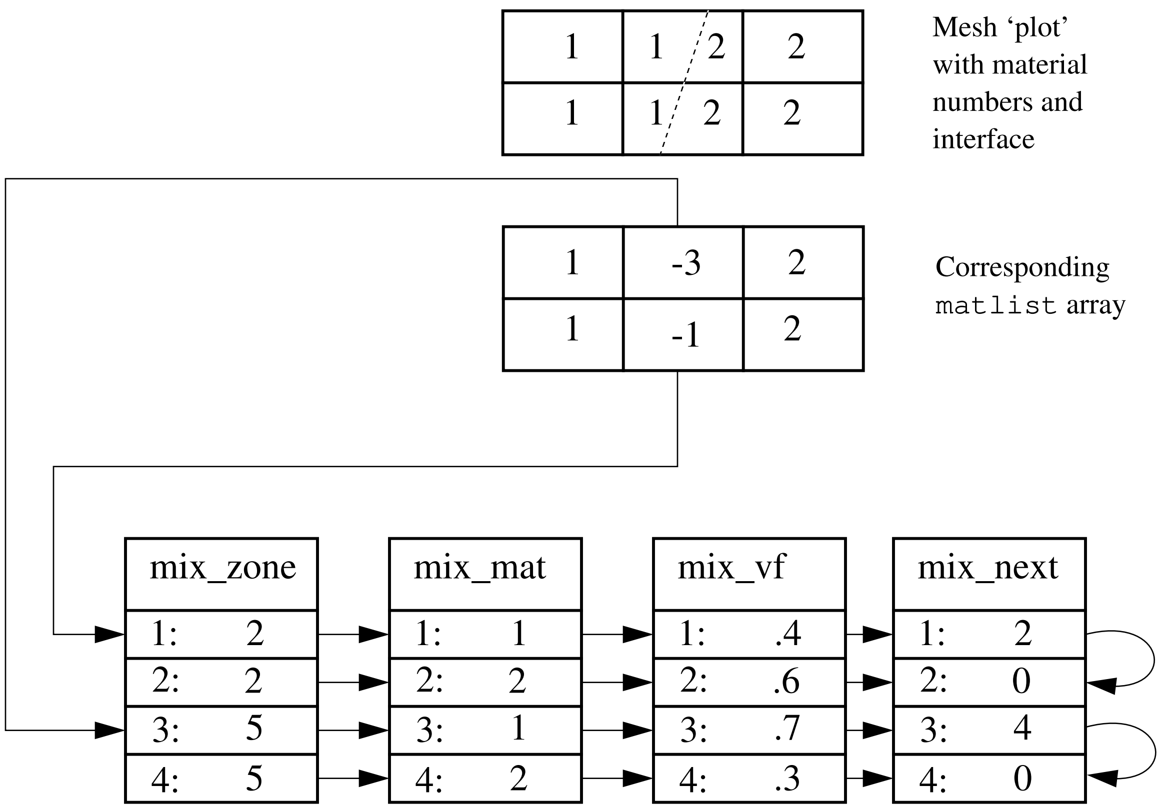


Figure 2-1: Example using mixed data arrays for representing material information

DBPutMatspecies()—Write a material species data object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutMatspecies(DBfile *dbfile, char *name, char *matname,
                    int nmat, int nmatspec[], int speclist[],
                    int dims[], int ndims, int nspecies_mf,
                    float species_mf[], int mix_list[],
                    int mixlen, int datatype, DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the material species data object.
`matname`	Name of the material object with which the material species object is associated.
`nmat`	Number of materials.
`nmatspec`	Array of length `nmat` containing the number of material species associated with each material.
`speclist`	Array of dimension `ndims * dims` of indices into the `species_mf` array. Each entry corresponds to one zone. A positive value is the index to the mass fractions of a clean zone's material species. A negative value means that the zone is a mixed zone and that the array `mix_list` contains the index to the species mass fractions.
`dims`	Array of length `ndims` that defines the length of the `speclist` array.
`ndims`	Number of dimensions in the `speclist` array.
`nspecies_mf`	Number of material species mass fractions.
`species_mf`	Array of length `nspecies_mf` containing mass fractions of the material species.
`mix_list`	Array of length `mixlen` containing indices into the `species_mf` array. These are used for mixed zones.
`mixlen`	Length of the `mix_list` array.
`datatype`	The datatype of the mass fraction data in `species_mf`. One of the predefined SILO data types.
`optlist`	Pointer to an option list structure containing additional information to be included in the object written into the SILO file. Use a NULL if there are no options.

Returns:

DBPutMatspecies() returns zero on success and -1 on failure.

Description:

The DBPutMatspecies() function writes a material species data object into a SILO file. The minimum required information for a material species data object is supplied via the standard arguments to this function. The optlist argument must be used for supplying any information not requested through the standard arguments.

DBPutMultimat()—Write a multi-block material object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutMultimat(DBfile *dbfile, char *name, int nmat, char
                  *matnames[], DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the multi-material object.
`nmat`	Number of materials provided.
`matnames`	Array of length `nmat` containing pointers to the names of the materials to be associated with the multi-material object.
`optlist`	Pointer to an option list structure containing additional information to be included in the object written into the SILO file. Use a NULL if there are no options

Returns:

DBPutMultimat() returns zero on success and -1 on error.

Description:

The DBPutMultimat() function writes a multi-material object into a SILO file.

DBPutMultimesh()—Write a multi-block mesh object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutMultimesh(DBfile *dbfile, char *name, int nmesh,
                   char *meshnames[],int meshtypes[],
                   DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the multi-block mesh structure.
`nmesh`	Number of meshes provided.
`meshnames`	Array of length `nmesh` containing pointers to the names of each of the meshes.
`meshtypes`	Array of length `nmesh` containing the type of each mesh. One of the predefined types: DB_QUAD_RECT, DB_QUAD_CURV, DB_UCDMESH, DB_POINTMESH.
`optlist`	Pointer to an option list structure containing additional information to be included in the object written into the SILO file. Use a NULL if there are no options.

Returns:

DBPutMultimesh() returns zero on success and -1 on failure.

Description:

The DBPutMultimesh() function writes a multi-block mesh object into a SILO file. It accepts as input descriptions of the various sub-meshes (blocks) which are part of this mesh.

DBPutMultivar()—Write a multi-block variable object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutMultivar(DBfile *dbfile, char *name, int nvar, char
                    *varnames[], int vartypes[], DBoptlist
                    *optlist);
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the multi-block variable.
`nvar`	Number of variables associated with the multi-block variable.
`varnames`	Array of length `nvar` containing pointers to the names of the variables. These are variables written with DBPutPointvar(), DBPutQuadvar(), and DBPutUcd-var().
`vartypes`	Array of length `nvar` containing the types of the variables. Each entry must be one of the following: DB_POINTVAR, DB_QUADVAR, or DB_UCDVAR.
`optlist`	Pointer to an option list structure containing additional information to be included in the object written into the SILO file. Use a NULL if there are no options.

Returns:

DBPutMultivar() returns zero on success and -1 on failure.

Description:

The DBPutMultivar() function writes a multi-block variable object into a SILO file.

DBPutPointmesh()—Write a point mesh object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutPointmesh(DBfile *dbfile, char *name, int ndims,
                   float *coords[], int nels, int datatype,
                   DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the mesh.
`ndims`	Number of dimensions.
`coords`	Array of length `ndims` containing pointers to coordinate arrays.
`nels`	Number of elements (points) in mesh.
`datatype`	Datatype of the coordinate arrays. One of the predefined SILO data types.
`optlist`	Pointer to an option list structure containing additional information to be included in the mesh object written into the SILO file. Typically, this argument is NULL.

Returns:

DBPutPointmesh() returns zero on success and -1 on failure.

Description:

The DBPutPointmesh() function accepts pointers to the coordinate arrays and is responsible for writing the mesh into a point-mesh object in the SILO file.

A SILO point-mesh object contains all necessary information for describing a mesh. This includes the coordinate arrays, the number of dimensions (1,2,3,...) and the number of points.

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_XLABEL	char *	Character string defining the label associated with the X dimension.	NULL
DBOPT_YLABEL	char *	Character string defining the label associated with the Y dimension.	NULL
DBOPT_ZLABEL	char *	Character string defining the label associated with the Z dimension.	NULL

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_NSPACE	int	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_XUNITS	char *	Character string defining the units associated with the X dimension.	NULL
DBOPT_YUNITS	char *	Character string defining the units associated with the Y dimension.	NULL
DBOPT_ZUNITS	char *	Character string defining the units associated with the Z dimension.	NULL

DBPutPointvar()—Write a vector/tensor point variable object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutPointvar(DBfile *dbfile, char *name, char *meshname,
                  int nvars, float *vars[], int nels,
                  int datatype, DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the variable set.
`meshname`	Name of the associated point mesh.
`nvars`	Number of variables supplied in `vars` array.
`vars`	Array of length `nvars` containing pointers to value arrays.
`nels`	Number of elements (points) in variable.
`datatype`	Datatype of the value arrays. One of the predefined SILO data types.
`optlist`	Pointer to an option list structure containing additional information to be included in the variable object written into the SILO file. Typically, this argument is NULL.

Returns:

DBPutPointvar() returns zero on success and -1 on failure.

Description:

The DBPutPointvar() function accepts pointers to the value arrays and is responsible for writing the variables into a point-variable object in the SILO file.

A SILO point-variable object contains all necessary information for describing a variable associated with a point mesh. This includes the number of arrays, the datatype of the variable, and the number of points. This function should be used when writing vector or tensor quantities. Otherwise, it is more convenient to use DBPutPointvar1().

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_NSPACE	int	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0

DBPutPointvar1()—Write a scalar point variable object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutPointvar1(DBfile *dbfile, char *name, char *meshname,
                   float var[], int nels, int datatype,
                   DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the variable.
`meshname`	Name of the associated point mesh.
`var`	Array containing data values for this variable.
`nels`	Number of elements (points) in variable.
`datatype`	Datatype of the variable. One of the predefined SILO data types.
`optlist`	Pointer to an option list structure containing additional information to be included in the variable object written into the SILO file. Typically, this argument is NULL.

Returns:

DBPutPointvar1() returns zero on success and -1 on failure.

Description:

The DBPutPointvar1() function accepts a value array and is responsible for writing the variable into a point-variable object in the SILO file.

A SILO point-variable object contains all necessary information for describing a variable associated with a point mesh. This includes the number of arrays, the datatype of the variable, and the number of points. This function should be used when writing scalar quantities. To write vector or tensor quantities, one must use DBPutPointvar().

Notes:

The following table describes the options accepted by this function. See the section titled "Using the SILO Option Parameter" for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_NSPACE	int	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0

DBPutQuadmesh()—Write a quad mesh object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutQuadmesh(DBfile *dbfile, char *name, char *coordnames[],
                  float *coords[], int dims[], int ndims,
                  int datatype, int coordtype,
                  DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the mesh.
`coordnames`	Array of length `ndims` containing pointers to the names to be provided when writing out the coordinate arrays. These are implemented as labels in the SILO database (see `optlist` below).
`coords`	Array of length `ndims` containing pointers to the coordinate arrays.
`dims`	Array of length `ndims` describing the dimensionality of the mesh. Each value in the `dims` array indicates the number of nodes contained in the mesh along that dimension.
`ndims`	Number of dimensions.
`datatype`	Datatype of the coordinate arrays. One of the predefined SILO data types.
`coordtype`	Coordinate array type. One of the predefined types: DB_COLLINEAR or DB_NONCOLLINEAR. Collinear coordinate arrays are always one-dimensional, regardless of the dimensionality of the mesh; non-collinear arrays have the same dimensionality as the mesh.
`optlist`	Pointer to an option list structure containing additional information to be included in the mesh object written into the SILO file. Typically, this argument is NULL.

Returns:

DBPutQuadmesh() returns zero on success and -1 on failure.

Description:

The DBPutQuadmesh() function accepts pointers to the coordinate arrays and is responsible for writing the mesh into a quad-mesh object in the SILO file.

A SILO quad-mesh object contains all necessary information for describing a mesh. This includes the coordinate arrays, the rank of the mesh (1,2,3,...) and the type (collinear or non-collinear). In addition, other information is useful and is therefore optionally included (row-major indicator, time and cycle of mesh, offsets to ‘real’ zones, plus coordinate system type.)

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_FACETYPE	int	Zone face type. One of the predefined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_HI_OFFSET	int *	Array of length `ndims` which defines zero-origin offsets from the last node for the ending index along each dimension.	{0,0,...}
DBOPT_LO_OFFSET	int *	Array of `ndims` which defines zero-origin offsets from the first node for the starting index along each dimension.	{0,0,...}
DBOPT_XLABEL	char *	Character string defining the label associated with the X dimension.	NULL
DBOPT_YLABEL	char *	Character string defining the label associated with the Y dimension.	NULL
DBOPT_ZLABEL	char *	Character string defining the label associated with the Z dimension.	NULL
DBOPT_MAJORORDER	int	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_NSPACE	int	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_PLANAR	int	Planar value. One of: DB_AREA or DB_VOLUME.	DB_OTHER
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_XUNITS	char *	Character string defining the units associated with the X dimension.	NULL
DBOPT_YUNITS	char *	Character string defining the units associated with the Y dimension.	NULL
DBOPT_ZUNITS	char *	Character string defining the units associated with the Z dimension.	NULL

The options DB_LO_OFFSET and DB_HI_OFFSET should be used if the mesh being described uses the notion of “phoney” zones (i.e., some zones should be ignored.) For example, if a 2-D mesh had designated the first column and row, and the last two columns and rows as “phoney”, then we would use: lo_off = {1,1} and hi_off = {2,2}.

DBPutQuadvar()—Write a vector/tensor quad variable object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutQuadvar(DBfile *dbfile, char *name, char *meshname, int
                   nvars, char *varnames[], float *vars[], int
                   dims[], int ndims, float *mixvars[], int
                   mixlen, int datatype, int centering, DBoptlist
                   *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the variable.
`meshname`	Name of the mesh associated with this variable (written with DBPutQuadmesh() or DBPutUcdmesh()). If no association is to be made, this value should be NULL.
`nvars`	Number of sub-variables which comprise this variable. For a scalar array, this is one. If writing a vector quantity, however, this would be two for a 2-D vector and three for a 3-D vector.
`varnames`	Array of length `nvars` containing pointers to character strings defining the names associated with each sub-variable.
`vars`	Array of length `nvars` containing pointers to arrays defining the values associated with each subvariable
`dims`	Array of length `ndims` which describes the dimensionality of the variable. Each value in the `dims` array indicates the number of elements contained in the variable along that dimension.
`ndims`	Number of dimensions.
`mixvars`	Array of length `nvars` containing pointers to arrays defining the mixed-data values associated with each subvariable. If no mixed values are present, this should be NULL.
`mixlen`	Length of mixed data arrays, if provided.
`datatype`	Datatype of the variable. One of the predefined SILO data types.
`centering`	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
`optlist`	Pointer to an option list structure containing additional information to be included in the variable object written into the SILO file. Typically, this argument is NULL.

Returns:

DBPutQuadvar() returns zero on success and -1 on failure.

Description:

The DBPutQuadvar() function writes a variable associated with a quad mesh into a SILO file. Note that variables will be either node-centered or zone-centered. A quad-var object contains the variable values. Other information can also be included. This function is useful for writing vector and tensor fields, whereas the companion function, DBPutQuadvar1(), is appropriate for writing scalar fields.

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_FACETYPE	int	Zone face type. One of the predefined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_LABEL	char *	Character string defining the label associated with this variable.	NULL
DBOPT_MAJORORDER	int	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_UNITS	char *	Character string defining the units associated with this variable.	NULL

DBPutQuadvar1() — Write a scalar quad variable object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutQuadvar1(DBfile *dbfile, char *name, char *meshname,
                    float *var, int dims[], int ndims,
                    float *mixvar, int mixlen, int datatype, int
                    centering, DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the variable.
`meshname`	Name of the mesh associated with this variable (written with DBPutQuadmesh() or DBPutUcdmesh().) If no association is to be made, this value should be NULL.
`var`	Array defining the values associated with this variable.
`dims`	Array of length `ndims` which describes the dimensionality of the variable. Each value in the `dims` array indicates the number of elements contained in the variable along that dimension.
`ndims`	Number of dimensions.
`mixvar`	Array defining the mixed-data values associated with this variable. If no mixed values are present, this should be NULL.
`mixlen`	Length of mixed data arrays, if provided.
`datatype`	Datatype of sub-variables. One of the predefined SILO data types.
`centering`	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
`optlist`	Pointer to an option list structure containing additional information to be included in the variable object written into the SILO file. Typically, this argument is NULL.

Returns:

DBPutQuadvar1() returns zero on success and -1 on failure.

Description:

The DBPutQuadvar1() function writes a scalar variable associated with a quad mesh into a SILO file. Note that variables will be either node-centered or zone-centered. A quad-var object contains the variable values, plus the name of the associated quad-mesh. Other information can also be included. This function should be used for writing scalar fields, and its companion function, DBPutQuadvar(), should be used for writing vector and tensor fields.

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_FACETYPE	int	Zone face type. One of the predefined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_LABEL	char *	Character string defining the label associated with this variable.	NULL
DBOPT_MAJORORDER	int	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_UNITS	char *	Character string defining the units associated with this variable.	NULL

DBPutUcdmesh()—Write a UCD mesh object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutUcdmesh(DBfile *dbfile, char *name, int ndims,
                 char *coordnames[], float *coords[],
                 int nnodes, int nzones,
                 char *zonel_name, char *facel_name,
                 int datatype, DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the mesh.
`ndims`	Number of spatial dimensions represented by this UCD mesh.
`coordnames`	Array of length `ndims` containing pointers to the names to be provided when writing out the coordinate arrays.*Currently, the coordinate names are ignored by SILO, but they must be non-NULL.*
`coords`	Array of length `ndims` containing pointers to the coordinate arrays.
`nnodes`	Number of nodes in this UCD mesh.
`nzones`	Number of zones in this UCD mesh.
`zonel_name`	Name of the zonelist structure associated with this variable [written with DBPutZonelist()]. If no association is to be made, this value should be NULL.
`facel_name`	Name of the facelist structure associated with this variable [written with DBPut-Facelist()]. If no association is to be made, this value should be NULL.
`datatype`	Datatype of the coordinate arrays. One of the predefined SILO data types.
`optlist`	Pointer to an option list structure containing additional information to be included in the mesh object written into the SILO file. See the table below for the valid options for this function. If no options are to be provided, use NULL for this argument.

Returns:

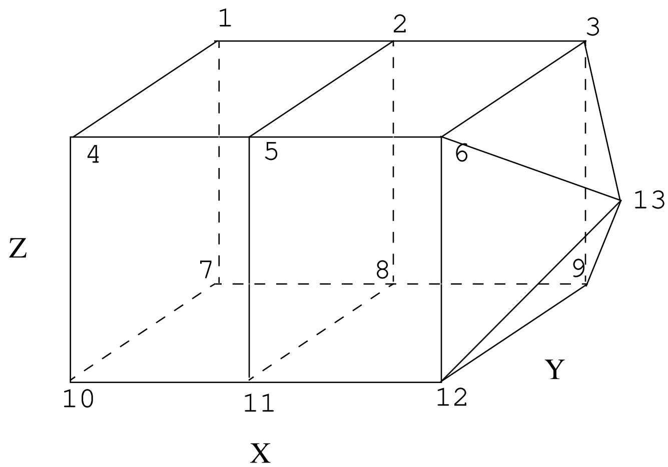
DBPutUcdmesh() returns zero on success and -1 on failure.

Description:

The DBPutUcdmesh() function accepts pointers to the coordinate arrays and is responsible for writing the mesh into a UCD mesh object in the SILO file.

A SILO UCD mesh object contains all necessary information for describing a mesh. This includes the coordinate arrays, the rank of the mesh (1,2,3,...) and the type (collinear or non-collinear.) In addition, other information is useful and is therefore included (time and cycle of mesh, plus coordinate system type).

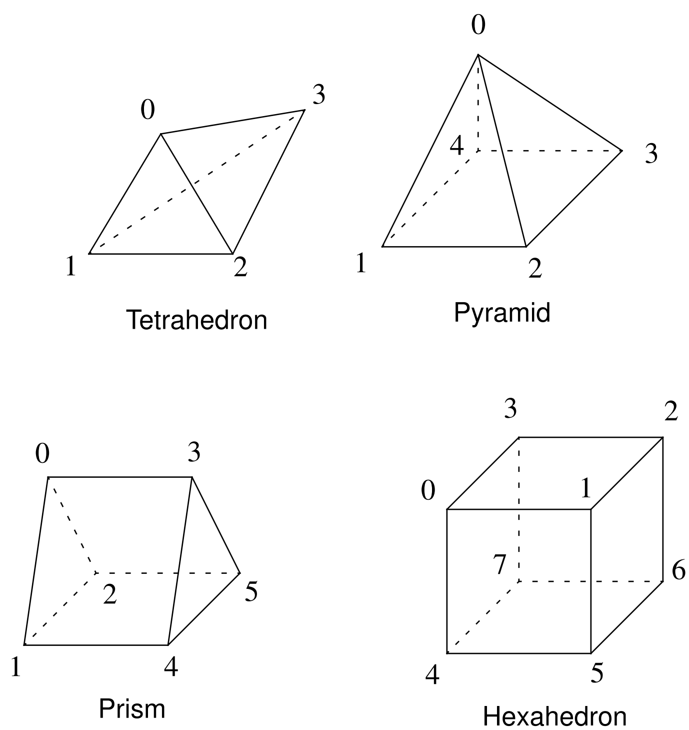
Notes:




```
nnodes     = 13
nzones     = 3
nzshapes   = 2
lznodelist = 2*8 + 1*5 = 21 zone nodes
nfaces     = 13 external faces
nfshapes   = 2 external face shapes
nftypes    = 0
lfnodelist = 9*4 + 4*3 = 48 external face nodes

fnodelist = { 1,2,8,7 external face nodelist
              2,3,9,8,
              8,9,12,11,
              5,6,12,11,...}

fshapesize = {4,3} external face shape sizes
fshapecnt = {9,4} external face shape counts
fzoneno    = {1,2,2,2,...}external face zone nos

znodelist = { 1,4,5,2,7,10,11,8, zone nodelist
              2,5,6,3,8,11,12,9,
              13,3,6,12,9}
zshapesize = {8,5} zone shape sizes
zshapecnt = {2,1} zone shape counts
x = {0,1,2,0,1,2,0,1,2,0,1,2,3}
y = {1,1,1,0,0,0,1,1,1,0,0,0,.5}
z = {1,1,1,1,1,1,0,0,0,0,0,0,.5}
```

Figure 2-2: Example usage of UCD zonelist and *external* facelist variables.

The order in which nodes are defined in the zonelist is important, especially for 3D cells. Nodes defining a 2D cell should be supplied in either clockwise or counterclockwise order around the cell. Nodes defining a 3D cell should be supplied in the order illustrated below.



The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_EDGELIST	int	Object ID for edgelist structure	0
DBOPT_FACETYPE	int	Zone face type. One of the predefined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_XLABEL	char *	Character string defining the label associated with the X dimension.	NULL

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_YLABEL	char *	Character string defining the label associated with the Y dimension.	NULL
DBOPT_ZLABEL	char *	Character string defining the label associated with the Z dimension.	NULL
DBOPT_NSPACE	int	Number of spatial dimensions used by this mesh.	`ndims`
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_PLANAR	int	Planar value. One of: DB_AREA or DB_VOLUME.	DB_NONE
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_XUNITS	char *	Character string defining the units associated with the X dimension.	NULL
DBOPT_YUNITS	char *	Character string defining the units associated with the Y dimension.	NULL
DBOPT_ZUNITS	char *	Character string defining the units associated with the Z dimension.	NULL

DBPutUcdvar()—Write a vector/tensor UCD variable object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutUcdvar(DBfile *dbfile, char *name, char *meshname,
                    int nvars, char *varnames[], float *vars[],
                    int nels, float *mixvars[], int mixlen,
                    int datatype, int centering,
                    DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the variable.
`meshname`	Name of the mesh associated with this variable [written with DBPutUcdmesh()]. If no association is to be made, this value should be NULL.
`nvars`	Number of sub-variables which comprise this variable. For a scalar array, this is one. If writing a vector quantity, however, this would be two for a 2-D vector and three for a 3-D vector.
`varnames`	Array of length `nvars` containing pointers to character strings defining the names associated with each subvariable.
`vars`	Array of length `nvars` containing pointers to arrays defining the values associated with each subvariable.
`nels`	Number of elements in this variable.
`mixvars`	Array of length `nvars` containing pointers to arrays defining the mixed-data values associated with each subvariable. If no mixed values are present, this should be NULL.
`mixlen`	Length of mixed data arrays (i.e., `mixvars`).
`datatype`	Datatype of sub-variables. One of the predefined SILO data types.
`centering`	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
`optlist`	Pointer to an option list structure containing additional information to be included in the variable object written into the SILO file. See the table below for the valid options for this function. If no options are to be provided, use NULL for this argument.

Returns:

DBPutUcdvar() returns zero on success and -1 on failure.

Description:

The DBPutUcdvar() function writes a variable associated with an UCD mesh into a SILO file. Note that variables will be either node-centered or zone-centered. Other information can also be included. This function is useful for writing vector and tensor fields, whereas the companion function, DBPutUcdvar1(), is appropriate for writing scalar fields.

Notes:

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_LABEL	char *	Character strings defining the label associated with this variable.	NULL
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_UNITS	char *	Character string defining the units associated with this variable.	NULL

`DBPutUcdvar1()`—Write a scalar UCD variable object into a SILO file.

Synopsis:

```
int DBPutUcdvar1 (DBfile *dbfile, char *name, char *meshname,
                    float *var, int nels, float *mixvar,
                    int mixlen, int datatype, int centering,
                    DBoptlist *optlist)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the variable.
`meshname`	Name of the mesh associated with this variable (written with either DBPutUcdmesh() or dbputum().) If no association is to be made, this value should be NULL.
`var`	Array of length `nels` containing the values associated with this variable.
`nels`	Number of elements in this variable.
`mixvar`	Array of length `mixlen` containing the mixed-data values associated with this variable. If `mixlen` is zero, this value is ignored.
`mixlen`	Length of mixvar array. If zero, no mixed data is present.
`datatype`	Datatype of variable. One of the predefined SILO data types.
`centering`	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
`optlist`	Pointer to an option list structure containing additional information to be included in the variable object written into the SILO file. See the table below for the valid options for this function. If no options are to be provided, use NULL for this argument.

Returns:

DBPutUcdvar1() returns zero on success and -1 on failure.

Description:

DBPutUcdvar1() writes a variable associated with an UCD mesh into a SILO file. Note that variables will be either node-centered or zone-centered. Other information can also be included. This function is useful for writing scalar fields, whereas the companion function, DBPutUcdvar(), is appropriate for writing vector and tensor fields.

Notes:

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_LABEL	char *	Character strings defining the label associated with this variable.	NULL
DBOPT_ORIGIN	int	Origin for arrays. Zero or one.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_UNITS	char *	Character string defining the units associated with this variable.	NULL

DBPutZonelist()—Write a zonelist object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBPutZonelist (DBfile *dbfile, char *name, int nzones,
                   int ndims, int nodelist[], int lnodelist,
                   int origin, int shapesize[], int shapecnt[],
                   int nshapes)
```

Arguments:

`dbfile`	Database file pointer.
`name`	Name of the zonelist structure.
`nzones`	Number of zones in associated mesh.
`ndims`	Number of spatial dimensions represented by associated mesh.
`nodelist`	Array of length `lnodelist` containing node indices describing mesh zones.
`lnodelist`	Length of nodelist array.
`origin`	Origin for indices in the nodelist array. Should be zero or one.
`shapesize`	Array of length `nshapes` containing the number of nodes used by each zone shape.
`shapecnt`	Array of length `nshapes` containing the number of zones having each shape.
`nshapes`	Number of zone shapes.

Returns:

DBPutZonelist() returns zero on success or -1 on failure.

Description:

The DBPutZonelist() function writes a zonelist object into a SILO file. The name assigned to this object can in turn be used as the `zonel_name` parameter to the DBPutUcdmesh() function.

Notes:

See the write-up of DBPutUcdmesh() for a full description of the zonelist data structures.

`DBReadAtt()`—Read an attribute value.

Synopsis:

```
#include <silo.h>
int DBReadAtt(DBfile *dbfile, char *varname, char *attname,
                    void *results)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable to which the attribute belongs.
`attname`	Name of the attribute.
`results`	Pointer to memory where attribute value should be stored.

Returns:

DBReadAtt() returns zero on success, and -1 on failure.

Description:

The DBReadAtt() function reads the given attribute value into the provided space.

Notes:

See DBGetAtt() for a memory-allocating version of this function.

DBReadVar()—Read a simple SILO variable.

Synopsis:

```
#include <silo.h>
int DBReadVar(DBfile *dbfile, char *varname, void *result)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.
`result`	Pointer to memory in which variable should be read. It is up to the application to provide sufficient space in which to read the variable.

Returns:

DBReadVar() returns zero on success and -1 on failure.

Description:

The DBReadVar() function reads a SILO variable into the given space.

Notes:

See DBGetVar() for a memory-allocating version of this function.

`DBReadVar1()`—Read one element from a simple SILO variable.

Synopsis:

```
#include <silo.h>
int DBReadVar1(DBfile *dbfile, char *varname, int offset,
                    void *result)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.
`offset`	Offset of one element to read.
`result`	Pointer to memory in which element should be read. It is up to the application to provide sufficient space in which to read the element.

Returns:

DBReadVar1() returns zero on success and -1 on failure.

Description:

The DBReadVar1() function reads one element from a SILO variable into the provided space.

`DBReadVarSlice()`—Read a (hyper)slab of data from a SILO variable.

Synopsis:

```
#include <silo.h>
int DBReadVarSlice (DBfile *dbfile, char *varname, int *offset,
                    int *length, int *stride, int ndims, void
                    *result)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.
`offset`	Array of length `ndims` of offsets in each dimension of the variable. This is the 0-origin position from which to begin reading the slice.
`length`	Array of length `ndims` of lengths of data in each dimension to read from the variable. All lengths must be positive.
`stride`	Array of length `ndims` of stride steps in each dimension. If no striding is desired, zeroes should be passed in this array.
`ndims`	Number of dimensions in the variable.
`result`	Pointer to location where the slice is to be written. It is up to the application to provide sufficient space in which to read the variable.

Returns:

DBReadVarSlice() returns zero on success and -1 on failure.

Description:

The DBReadVarSlice() function reads a slab of data from a SILO variable into a location provided in the `result` pointer. Any hyperslab of data may be read.

Note that the minimum `length` value is 1 and the minimum `stride` value is zero.

A one-dimensional array slice:

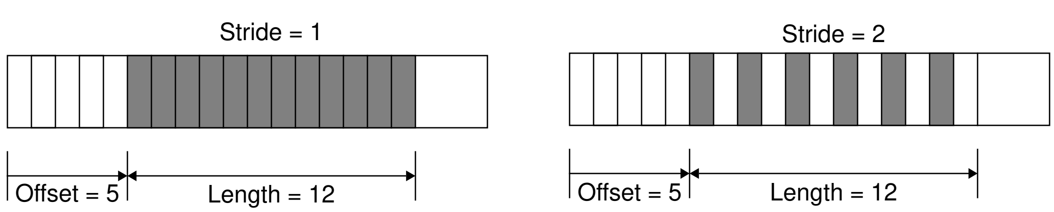


Figure 2-3: Array slice

DBSetDir()—Set the current directory within the SILO database.

Synopsis:

```
#include <silo.h>
int DBSetDir(DBfile *dbfile, char *pathname)
```

Arguments:

`dbfile`	Database file pointer.
`pathname`	Path name of the directory. This can be either an absolute or relative path name.

Returns:

DBSetDir() returns zero on success and -1 on failure.

Description:

The DBSetDir() function sets the current directory within the given SILO database.

DBShowErrors()—Set the error reporting mode.

Synopsis:

```
#include <silo.h>
void DBShowErrors(int level, void (*func)(char*))
```

Arguments:

`level`	Error reporting level. One of DB_ALL, DB_ABORT, DB_TOP, or DB_NONE.
`func`	Function pointer to an error-handling function.

Returns:

DBShowErrors() returns nothing (void). It cannot fail.

Description:

The DBShowErrors() function sets the level of error reporting done by SILO when it encounters an error. The following table describes the action taken upon error for different values of `level`:

Error level value	Error action
DB_ALL	Show all errors, beginning with the (possibly internal) routine that first detected the error and continuing up the call stack to the application.
DB_ABORT	Same as DB_ALL except abort() is called after the error message is printed.
DB_TOP	(Default) Only the top-level SILO functions issue error messages.
DB_NONE	The library does not handle error messages. The application is responsible for checking the return values of the SILO functions and handling the error.

For more information, see “Error Handling” on page 2-1.

`DBWrite()`—Write a simple SILO variable.

Synopsis:

```
#include <silo.h>
int DBWrite (DBfile *dbfile, char *varname, void *var, int *dims,
                   int ndims, int datatype)
```

Arguments:

`dbfile`	Database file pointer.
`varname`	Name of the variable.
`var`	Array defining the values associated with the variable.
`dims`	Array of length `ndims` which describes the dimensionality of the variable. Each value in the `dims` array indicates the number of elements contained in the variable along that dimension.
`ndims`	Number of dimensions.
`datatype`	Datatype of the variable. One of the predefined SILO data types.

Returns:

DBWrite() returns zero on success and -1 on failure.

Description:

The DBWrite() function writes a simple variable into a SILO file.

DBWriteComponent()—Add a variable component to an object and write the associated data.

Synopsis:

```
#include <silo.h>
int DBWriteComponent (DBfile *dbfile, DBobject *object, char
                     *compname, char *prefix, char *datatype, void
                     *var, int nd, long *count)
```

Arguments:

`dbfile`	Database file pointer.
`object`	Pointer to the object.
`compname`	Component name.
`prefix`	Path name prefix of the object.
`datatype`	Data type of the component's data. One of: "short", "integer", "long", "float", "double", "char".
`var`	Pointer to the component's data.
`nd`	Number of dimensions of the component.
`count`	An array of length `nd` containing the length of the component in each of its dimensions.

Returns:

DBWriteComponent() returns zero on success and -1 on failure.

Description:

The DBWriteComponent() function adds a component to an existing object and also writes the component's data to a SILO file.

`DBWriteObject()`—Write an object into a SILO file.

Synopsis:

```
#include <silo.h>
int DBWriteObject (DBfile *dbfile, DBobject *object)
```

Arguments:

`dbfile`	Database file pointer.
`object`	Object created with DBMakeObject() and populated with DBAddFltComponent(), DBAddIntComponent(), DBAddStrComponent(), and DBAddVarComponent.

Returns:

DBWriteObject() returns zero on success and -1 on failure.

Description:

The DBWriteObject() function writes an object into a SILO file. This is a user-defined object that consists of various components. They are used when the basic SILO structures are not sufficient.

2.2. Summary of C Interface

```
#include <silo.h>
int              DBAddFltComponent (Object *object, char
                            *compname, double f);
int              DBAddIntComponent (Object *object, char
                            *compname, int i);
int              DBAddOption (DBoptlist *optlist, int option, void
                            *value);
int              DBAddStrComponent (Object *object, char
                            *compname, char *s);
int              DBAddVarComponent (DBObject *object, char
                            *compname, char *vardata);
DBcompoundarray  *DBAllocCompoundarray (void);
DBedgelist       *DBAllocEdgelist (void);
DBfacelist       *DBAllocFacelist (void);
DBmaterial       *DBAllocMaterial (void);
DBmatspecies     *DBAllocMatspecies (void);
DBmeshvar        *DBAllocMeshvar (void);
DBmultimesh      *DBAllocMultimesh (void);
DBmultivar       *DBAllocMultivar (void);
DBpointmesh      *DBAllocPointmesh (void);
DBquadmesh       *DBAllocQuadmesh (void);
DBquadvar        *DBAllocQuadvar (void);
DBucdmesh        *DBAllocUcdmesh (void);
DBucdvar         *DBAllocUcdvar (void);
DBzonelist       *DBAllocZonelist (void);
DBfacelist       *DBCalcExternalFacelist (int znodelist[], int
                            nnodes,
                            int origin, int zshapesize[],
                            int zshapecnt[], int nzshapes,
                            int matlist[], int bnd_method);
int              DBClearObject (DBobject *object);
int              DBClearOptlist (DBoptlist *optlist);
int              DBClose (DBfile *dbfile);
int              DBContinue (DBfile *dbfile);
DBfile           *DBCreate (char *pathname, int mode, int target,
                            char *fileinfo, int filetype);
char             *DBErrFunc (void);
int              DBErrno (void);
char             *DBErrString (void);
int              DBFreeCompoundarray (DBcompoundarray *x);
int              DBFreeEdgelist (DBedgelist *x);
int              DBFreeFacelist (DBfacelist *x);
int              DBFreeMaterial (DBmaterial *x);
int              DBFreeMatspecies (DBmatspecies *x);
int              DBFreeMeshvar (DBmeshvar *x);
int              DBFreeMultimesh (DBmultimesh *x);
int              DBFreeMultivar (DBmultivar *x);
int              DBFreeObject (DBobject *x);
```

```
int              DBFreeOptlist (DBoptlist *optlist);
int              DBFreePointmesh (DBpointmesh *x);
int              DBFreeQuadmesh (DBquadmesh *x);
int              DBFreeQuadvar (DBquadvar *x);
int              DBFreeUcdmesh (DBucdmesh *x);
int              DBFreeUcdvar (DBucdvar *x);
int              DBFreeZonelist (DBzonelist *x);
void             *DBGetAtt (DBfile *dbfile, char *varname, char
                           *attname);
void             *DBGetComponent (DBfile *dbfile, char *objname,
                           char *compname);
DBcompoundarray  *DBGetCompoundarray (DBfile *dbfile, char
                           *arrayname);
DBcurve          *DBGetCurve (DBfile *dbfile, char *curvename);
int              DBGetDir (DBfile *dbfile, char *dirname);
DBmaterial       *DBGetMaterial (DBfile *dbfile, char *mat_name);
DBmatspecies     *DBGetMatspecies (DBfile *dbfile, char *ms_name);
DBmultimesh      *DBGetMultimesh (DBfile *dbfile, char *meshname);
DBPointmesh      *DBGetPointmesh (DBfile *dbfile, char *meshname);
DBmeshvar        *DBGetPointvar (DBfile *dbfile, char *varname);
DBquadmesh       *DBGetQuadmesh (DBfile *dbfile, char *meshname);
DBquadvar        *DBGetQuadvar (DBfile *dbfile, char *varname);
Dbtoc            *DBGetToc (DBfile *dbfile);
DBucdmesh        *DBGetUcdmesh (DBfile *dbfile, char *meshname);
DBucdvar         *DBGetUcdvar (DBfile *dbfile, char *varname);
void             *DBGetVar (DBfile *dbfile, char *varname);
int              DBGetVarByteLength (DBfile *dbfile, char
                           *varname);
int              DBGetVarLength (DBfile *dbfile, char *varname);
int              DBInqCompoundarray (DBfile *dbfile, char *name,
                           char *elemnames[], int *elemlengths,
                           int nelems, int nvalues, int
                           datatype);
int              DBInqMeshname (DBfile *dbfile, char *varname,
                           char *meshname);
int              DBInqMeshtype (DBfile *dbfile, char *meshname);
DBobject         *DBMakeObject (char *objname, int objtype, int
                           maxcomps);
DBoptlist        *DBMakeOptlist (int maxopts);
int              DBMkDir (DBfile *dbfile, char *dirname);
DBfile           *DBOpen (char *name, int type);
int              DBPause (DBfile *dbfile);
int              DBPutCompoundarray (DBfile *dbfile, char *name,
                           char *elemnames[], int *elemlengths,
                           int nelems, void *values, int nvalues,
                           int datatype, DBoptlist *optlist);
int              DBPutFacelist (DBfile *dbfile, char *name, int
                           nfaces, int ndims, int nodelist[], int
                           lnodelist, int origin, int zoneno[],
                           int shapesize[], int shapecnt[], int
```

```
                    nshapes, int types[], int typelist[],
                    int ntypes);
int     DBPutMaterial (DBfile *dbfile, char *name, char
                    *meshname, int nmat, int matnos[], int
                    matlist[], int dims[], int ndims, int
                    mix_next[],int mix_mat[], int
                    mix_zone[], float mix_vf[], int
                    mixlen, int datatype, DBoptlist
                    *optlist);
int     DBPutMatspecies (DBfile *dbfile, char *name, char
                    *matname, int nspecies_mf, float
                    species_mf[], int nmatspec[], int
                    nmat, int datatype, int speclist[],
                    int dims[], int ndims, int mix_list[],
                    int mixlen, DBoptlist *optlist);
int     DBPutMultimat (DBfile *dbfile, char *name, int
                    nmat, char *matnames[], DBoptlist
                    *optlist);
int     DBPutMultimesh (DBfile *dbfile, char *name, int
                    nmesh, char *meshnames[], int
                    meshtypes[], DBoptlist *optlist);
int     DBPutMultivar (DBfile *dbfile, char *name, char
                    *varnames[], int vartypes[], DBoptlist
                    *optlist);
int     DBPutPointmesh (DBfile *dbfile, char *name, int
                    ndims, float *coords[], int nels, int
                    datatype, DBoptlist *optlist)
int     DBPutPointvar (DBfile *dbfile, char *name, char
                    *meshname, int nvars, float *vars[],
                    int nels, int datatype, DBoptlist
                    *optlist);
int     DBPutPointvar1 (DBfile *dbfile, char *name, char
                    *meshname, float var[], int nels, int
                    datatype, DBoptlist *optlist);
int     DBPutQuadmesh (DBfile *dbfile, char *name, char
                    *coordnames[], float *coords[], int
                    dims[], int ndims, int datatype, int
                    coordtype, DBoptlist *optlist)
int     DBPutQuadvar (DBfile *dbfile, char *name, char
                    *meshname,int nvars, char *varnames[],
                    float *vars[], int dims[], int ndims,
                    float *mixvars[], int mixlen, int
                    datatype, int centering, DBoptlist
                    *optlist)
int     DBPutQuadvar1 (DBfile *dbfile, char *name, char
                    *meshname, float *var, int dims[], int
                    ndims, float *mixvar, int mixlen, int
                    datatype, int centering, DBoptlist
                    *optlist)
int     DBPutUcdmesh (DBfile *dbfile, char *name, int
```

```
                ndims, char *coordnames[], float
                *coords[], int nnodes, int nzones,
                char *zonel_name, char *facel_name,
                int datatype, DBoptlist *optlist)
int         DBPutUcdvar (DBfile *dbfile, char *name, char
                *meshname, int nvars, char
                *varnames[], float *vars[], int nels,
                float *mixvars[], int mixlen, int
                datatype, int centering, DBoptlist
                *optlist)
int         DBPutUcdvar1 (DBfile *dbfile, char *name, char
                *meshname, float *var, int nels, float
                *mixvar, int mixlen, int datatype, int
                centering, DBoptlist *optlist);
int         DBPutZonelist (DBfile *dbfile, char *name, int
                nzones, int ndims, int nodelist[], int
                lnodelist, int origin, int
                shapesize[], int shapecnt[], int
                nshapes);
int         DBReadAtt (DBfile *dbfile, char *varname, char
                *attname, void *results);
int         DBReadVar (DBfile *dbfile, char *varname, void
                *result);
int         DBReadVar1 (DBfile *dbfile, char *varname, int
                offset, void *result);
int         DBReadVarSlice (DBfile *dbfile, char *varname,
                int *offset, int *length, int *stride,
                int ndims, void *result);
int         DBSetDir (DBfile *dbfile, char *pathname);
void        DBShowErrors (int level, void (*func)(char*));
int         DBWrite (DBfile *dbfile, char *varname, void
                *var, int *dims, int ndims, int
                datatype);
int         DBWriteComponent (DBfile *dbfile, DBobject
                *object, char *compname, char *prefix,
                char *datatype, void *var, int nd,
                long *count);
int         DBWriteObject (DBfile *dbfile, DBobject *object);
```

Chapter 3

Fortran Functions

3.1. Fortran Interface

The following section contains the Fortran function descriptions for the SILO input and output packages. The interface supports both quadrilateral and UCD-based data. Currently, C-callable functions exist for all routines, but Fortran-callable functions exist for only a portion of the routines. The functions are arranged in alphabetical order.

3.1.1. Error Handling

SILO has an error-reporting function dbshowerrors that allows the programmer to tailor the reporting of errors. This function takes as its argument an error level, which is one of the following values:

Error level value	Error action
DB_ALL	Show all errors, beginning with the (possibly internal) routine that first detected the error and continuing up the call stack to the application.
DB_ABORT	Same as DB_ALL except abort() is called after the error message is printed.
DB_TOP	(Default) Only the top-level SILO functions issue error messages.
DB_NONE	The library does not handle error messages. The application is responsible for checking the return values of the SILO functions and handling the error.

3.1.2. Optional Arguments

The functions described below have optional arguments. By optional, it is meant that a dummy value can be supplied instead of an actual value. An argument to a Fortran function, which the user does not want to provide, and which is documented as optional, should be replaced with the parameter DB_F77NULL, which is defined in the file `silo.inc`.

3.1.3. Using the SILO Option Parameter

Many of the functions take as one of their arguments a list of option-name/option-value pairs. In this way, additional information can be passed to a function without having to change the function's interface. The following sequence of function declarations outlines the procedure for creating and populating such a list:

```
integer function dbmkoptlist(    ! Create a list:
          maxopts,               ! maximum list length
          optlist_id             ! list identifier
          )

integer function dbaddiopt (     ! Add an integer option
                                 ! to the list:
          optlist_id,            ! the list
          option_id,             ! the option
          int_value              ! the option's integer
                                 ! value
          )
```

There also are functions for adding real and character option values to a list.

3.1.4. Fortran Calling Sequence

The functions in the SILO output package should be called in a particular order. Start by creating a SILO file, with `dbcreate()`, create any necessary directories, then call the remaining routines as needed for writing out the mesh, material data, and any physics variables associated with the mesh.

Schematically, your program should look something like this:

```
dbcreate

dbmkdir
dbsetdir
     dbputqm
     dbputqv1
     dbputqv1
     dbputqv1
      . . .
dbsetdir

dbmkdir
dbsetdir
     dbputzl
     dbputfl
     dbputum
     dbputmat
```

```
     dbputuv1
 . . .
dbsetdir

dbclose
```

I

dbaddopt()—Add an option to an option list.

Synopsis:

```
include "silo.inc"
integer function dbaddcopt (optlist_id, option, cvalue, lcvalue)
integer function dbadddopt (optlist_id, option, dvalue)
integer function dbaddiopt (optlist_id, option, ivalue)
integer function dbaddropt (optlist_id, option, rvalue)

integer ivalue, optlist_id, option, lcvalue
double precision dvalue
real rvalue
character*(*) cvalue
```

Arguments:

`optlist_id`	Identifier returned from a previous call to dbmkoptlist().
`option`	Option definition. One of the predefined values described in the table in the notes section of each command which accepts an option list. Also see Table 8.
`ivalue`	Integer value associated with `option`.
`dvalue`	Double precision value associated with option.
`rvalue`	Real value associated with `option`.
`cvalue`	Character value associated with `option`.
`lcvalue`	Length of the `cvalue` variable parameter in characters.

Returns:

These functions return zero on success and -1 on failure.

Description:

The dbaddopt functions add an option/value pair to an option list. Several of the output functions accept option lists to provide information of an ancillary nature. This Fortran interface is split into multiple functions, one for each data type. Use the function appropriate for the given option type.

dbcalcfl()—Calculate an external facelist for a UCD mesh.

Synopsis:

```
include "silo.inc"
integer dbcalcfl (znodelist, nnodes, origin, zshapesize,
                  zshapecnt, nzshapes, matlist, bnd_method, id)

integer zonelist(*), nnodes, origin, zshapesize(*)
integer zshapecnt(*), nzshapes, matlist(*), bnd_method, id
```

Arguments:

`znodelist`	Array containing node indices describing mesh zones.
`nnodes`	Number of nodes in associated mesh.
`origin`	Origin for indices in the `znodelist` array. Should be zero or one.
`zshapesize`	Array of length `nzshapes` containing the number of nodes used by each zone shape.
`zshapecnt`	Array of length `nzshapes` containing the number of zones having each shape.
`nzshapes`	Number of zone shapes.
`matlist`	Array providing material numbers for each zone (else DB_F77NULL).
`bnd_method`	Method to use for calculating external faces. See description below.
`id`	Returned facelist identifier. (This should not be relied upon, as the use of facelist identifiers is no longer supported.)

Returns:

dbcalcfl() returns zero on success and -1 on failure.

Description:

The dbcalcfl() function calculates an external facelist from the zonelist and zone information describing a UCD mesh. It returns the object identifier for this object. The facelist should be written into the mesh with the dbputfl() function. The calculation of the external facelist is controlled by the `bnd_method` parameter as defined in the table below:

bnd_method	Meaning
0	Do not use material boundaries when computing external faces. The matlist parameter can be replaced with DB_F77NULL.
1	In addition to true external faces, include faces on material boundaries between two clean zones. The matlist parameter must be provided.

bnd_method	Meaning
2	In addition to true external faces, include faces on material boundaries between two clean zones and between one clean and one mixed zone (only the face from the clean zone will be included.) The matlist parameter must be provided.
4	In addition to true external faces, include faces on material boundaries between two clean zones and between one clean and one mixed zone (both faces will be included in this case.) The matlist parameter must be provided.

Notes:

This function currently only works for quad-shaped nodes.

dbclose()—Close a SILO database.

Synopsis:

```
include "silo.inc"
integer function dbclose (dbid)

integer dbid
```

Arguments:

`dbid`	Database identifier.

Returns:

dbclose() returns zero on success and -1 on failure.

Description:

The dbclose() function closes a SILO database.

`dbcreate()`—Create a SILO output file.

Synopsis:

```
include "silo.inc"
integer function dbcreate (pathname, lpathname, mode, target,
                 fileinfo, lfileinfo, filetype, dbid)

character*(*) pathname, fileinfo
integer lpathname, mode, target, lfileinfo, filetype, dbid
```

Arguments:

`pathname`	Pathname of the file to create. This can be either an absolute or relative path.
`lpathname`	Length of the `pathname` parameter in characters.
`mode`	Creation mode. One of the predefined SILO modes: DB_CLOBBER or DB_NOCLOBBER.
`target`	Destination file format. One of the predefined types: DB_LOCAL, DB_SUN3, DB_SUN4, DB_SGI, DB_RS6000, or DB_CRAY.
`fileinfo`	Character string containing descriptive information about the file's contents. This information is usually printed by applications when this file is opened. If no such information is needed, send DB_F77NULL for this argument.
`lfileinfo`	Length of the `fileinfo` parameter in characters.
`filetype`	Destination file type. Currently only one type is supported: DB_PDB.
`dbid`	Returned database identifier for this file.

Returns:

dbcreate() returns zero on success and -1 on failure.

Description:

The dbcreate() function creates a SILO file and initializes it for writing data.

Notes:

The underlying database library (PDBLib) supports the concept of targeting output files. That is, a Sun IEEE file can be created on the Cray, and vice versa. If creating files on a mainframe or other powerful computer, it is best to target the file for the machine where the file will be processed. Because of the extra time required to do the floating point conversions, however, one may wish to bypass the targeting function by providing DB_LOCAL as the target.

SILO currently creates only one kind of file, a PDB file. This PDB file contains some special structures for handling objects and directory hierarchies.

Note that regardless of what type of file is created, it can still be read on any machine.

`dbfgetca()`—Fast get of Compound Array object from SILO file.

Synopsis:

```
include "silo.inc"
integer function dbfgetca (dbid, name, lname, values, nvalues)

integer dbid, lname, nvalues
character*(*) name
real values(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the compound array.
`lname`	Length of the `name` parameter in characters.
`values`	Returned array of length `nvalues` containing the data values.
`nvalues`	Returned length of the `values` array.

Returns:

dbfgetca() returns zero on success and -1 on failure.

Description:

The dbfgetca() function reads in a compound array from the SILO database, and returns the values associated with it.

`dbfreeoptlist()`—Free memory associated with an option list.

Synopsis:

```
include "silo.inc"
integer function dbfreeoptlist (optlist_id)

integer optlist_id
```

Arguments:

`optlist_id` Identifier returned from a previous call to dbmkoptlist().

Returns:

dbfreeoptlist() returns zero on success and -1 on failure.

Description:

The dbfreeoptlist() function releases the memory associated with the given option list. The individual option values are not freed.

`dbgetca()`—Read a compound array from a SILO database.

Synopsis:

```
include "silo.inc"
integer function dbgetca (dbid, name, lname, enames, lenames,
                    elengths, nelems, values, nvalues, datatype)

integer dbid, lname, nelems, nvalues, datatype
integer lenames(*), elengths(*)
character*(*) name, enames
real values(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the compound array.
`lname`	Length of the `name` parameter in characters.
`enames`	Array with length determined by `lenames` containing the returned names of the elements. These names are stored sequentially in the array.
`lenames`	Returned array of length `nelems` containing the lengths of the names stored in the enames array.
`elengths`	Returned array of length `nelems` containing the lengths of the elements.
`nelems`	Number of elements.
`values`	Returned array of length `nvalues` containing the data values.
`nvalues`	Number of values.
`datatype`	The datatype of the values. One of the predefined SILO data types.

Returns:

dbgetca() returns zero on success and -1 on failure.

Description:

The dbgetca() function reads in a compound array from the SILO database, and returns the names and values associated with it.

dbinqca()—Inquire Compound Array attributes.

Synopsis:

```
include "silo.inc"
integer function dbinqca (dbid, name, lname, tlenames, nelems,
                   nvalues, datatype)

integer dbid, lname, tlenames, nelems, nvalues, datatype
character*(*) name
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the compound array.
`lname`	Length of the `name` parameter in characters.
`tlenames`	Returned sum of the lengths of the element names.
`nelems`	Returned number of array elements.
`nvalues`	Returned number of data values.
`datatype`	Datatype of the data values. One of the predefined SILO data types.

Returns:

dbinqca() returns zero on success and -1 on failure.

Description:

The dbinqca() function returns information about the Compound Array. It does not return the data values themselves; use dbgetca() instead.

`dbinqlen()`—Return the number of elements in a variable.

Synopsis:

```
integer function dbinqlen (dbid, varname, lvarname, len)

integer dbid, lvarname, len
character*(*) varname
```

Arguments:

`dbid`	Database identifier.
`varname`	Variable name.
`lvarname`	Length of the `varname` parameter in characters.
`len`	Length of the variable (number of elements).

Returns:

dbinqlen() returns zero on success and -1 on failure.

Description:

The dbinqlen() function returns the length of the requested variable in number of elements.

dbmkdir()—Create a new directory in the open SILO file.

Synopsis:

```
include "silo.inc"
integer function dbmkdir (dbid, dirname, ldirname, id)

integer dbid, ldirname, id
character*(*) dirname
```

Arguments:

`dbid`	Database identifier.
`dirname`	Name of the directory to create.
`ldirname`	Length of the `dirname` parameter in characters.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of directory identifiers is no longer supported.)

Returns:

dbmkdir() returns zero on success and -1 on failure.

Description:

The dbmkdir() function creates a new directory in the open SILO file as a child of the current directory (see dbsetdir()). The directory name may be an absolute path name similar to "`/dir/subdir`", or may be a relative path name similar to "`../../dir/subdir`".

`dbmkoptlist()`—Allocate an option list.

Synopsis:

```
include "silo.inc"
integer function dbmkoptlist (maxopts, optlist_id)

integer maxopts, optlist_id
```

Arguments:

`maxopts`	Maximum number of options needed for this option list.
`optlist_id`	Returned identifier for this option list.

Returns:

dbmkoptlist() returns zero on success and -1 on failure.

Description:

The dbmkoptlist() function allocates memory for an option list and initializes it. Use the function dbaddopt() to populate the option list structure, and dbfreeoptlist() to free it.

`dbopen()`—Open an existing SILO file.

Synopsis:

```
include "silo.inc"
integer function dbopen (name, lname, type, dbid)

integer type, lname, dbid
character*(*) name
```

Arguments:

`name`	Name of the file to open. Can be either an absolute or relative path.
`lname`	Length of the `name` parameter in characters.
`type`	The type of file to open. One of the predefined types: DB_SDX, DB_PDB, DB_TAURUS, or DB_UNKNOWN.
`dbid`	Returned database identifier.

Returns:

dbopen() returns zero on success and -1 on failure.

Description:

The dbopen() function opens an existing SILO file. If the file `type` if DB_UNKNOWN, SILO will guess at the file type, getting it right most of the time.

dbputca()—Write a Compound Array object into a SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputca (dbid, name, lname, enames, lenames,
                    elengths, nelems, values, nvalues, datatype,
                    optlist_id)

integer dbid, lname, nelems, nvalues, datatype, optlist_id
integer lenames(*), elengths(*)
character*(*) name
character*32 enames(*)
real values(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the compound array.
`lname`	Length of the name parameter in characters.
`enames`	Array of length `nelems` containing the names of the elements. This is an array of 32-character strings.
`lenames`	Array of length `nelems` containing the lengths of the `enames` strings.
`elengths`	Array of length `nelems` containing the lengths of the elements.
`nelems`	Number of simple array elements.
`values`	Array whose length is determined by `nelems` and `elengths` containing the values of the simple array elements.
`nvalues`	Total length of the `values` array.
`datatype`	Data type of the `values` array. One of the predefined SILO types.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of compound array identifiers is no longer supported.)

Returns:

dbputca() returns zero on success and -1 on failure.

Description:

The dbputca() function writes a compound array object into a SILO file. A compound array is an array whose elements are simple arrays. All of the simple arrays have elements of the same data type, and each have a name.

Often, an application will partition a block of memory into named pieces, but write the block to a database as a single entity. Fortran common blocks are used in this way. The compound array object is an abstraction of this partitioned memory block.

`dbputfl()`—Write a facelist object into a SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputfl (dbid, name, lname, nfaces, ndims,
                    nodelist, lnodelist, origin, zoneno,
                    shapesize, shapecnt, nshapes, types, typelist,
                    ntypes, id)

integer dbid, lname, nfaces, ndims, lnodelist, origin, nshapes
integer ntypes, id
character*(*) name
integer nodelist(*), shapesize(*), shapecnt(*)
integer zoneno(*), types(*), typelist(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the facelist structure.
`lname`	Length of the name parameter in characters.
`nfaces`	Number of external faces in associated mesh.
`ndims`	Number of spatial dimensions represented by the associated mesh.
`nodelist`	Array of length `lnodelist` containing node indices describing mesh faces.
`lnodelist`	Length of the `nodelist` array.
`origin`	Origin for indices in `nodelist` array. Either zero or one.
`zoneno`	Array of length `nfaces` containing the zone number from which each face came. Use DB_F77NULL for this parameter if zone numbering info is not wanted.
`shapesize`	Array of length `nshapes` containing the number of nodes used by each face shape (for 3-D meshes only).
`shapecnt`	Array of length `nshapes` containing the number of faces having each shape (for 3-D meshes only).
`nshapes`	Number of face shapes (for 3-D meshes only).
`types`	Array of length `nfaces` containing information about each face. This argument is ignored if `ntypes` is zero, or if this parameter is DB_F77NULL.
`typelist`	Array of length `ntypes` containing the identifiers for each type. This argument is ignored if `ntypes` is zero, or if this parameter is DB_F77NULL.
`ntypes`	Number of types, or zero if type information was not provided.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of facelist identifiers is no longer supported.)

Returns:

dbputfl() returns zero on success and -1 on failure.

Description:

The dbputfl() function writes a facelist object into the open SILO file. The name given to this object can in turn be used as a parameter to the dbputum() function.

Notes:

See the write-up of dbputum() for a full description of the facelist data structures. *Note that MeshTV expects this structure to contain descriptions of the external faces only.*

dbputmat()—Write a material data object into a SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputmat (dbid, name, lname, meshname, lmeshname,
                     nmat, matnos, matlist, dims, ndims, mix_next,
                     mix_mat, mix_zone, mix_vf, mixlen, datatype,
                     optlist_id, id)

integer dbid, lname, lmeshname, nmat, ndims, mixlen, datatype
integer optlist_id, id
integer matnos(*), matlist(*), dims(*), mix_next(*), mix_mat(*)
integer mix_zone(*)
character*(*) name
character*(*) meshname
real mix_vf(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the material data object.
`lname`	Length of the `name` parameter in characters.
`meshname`	Name of the mesh associated with this information.
`lmeshname`	Length of the `meshname` parameter in characters.
`nmat`	Number of materials.
`matnos`	Array of length `nmat` containing valid material numbers.
`matlist`	Array whose dimensions are defined by `dims` and `ndims`. It contains the material numbers for each single-material (non-mixed) zone, and indices into the mixed data arrays for each multi-material (mixed) zone. A negative value indicates a mixed zone, and its absolute value is used as an index into the mixed data arrays.
`dims`	Array of length `ndims` which defines the dimensionality of the `matlist` array.
`ndims`	Number of dimensions in `matlist` array.
`mix_next`	Array of length `mixlen` of indices into the mixed data arrays (one-origin).
`mix_mat`	Array of length `mixlen` of material numbers for the mixed zones.
`mix_zone`	Optional array of length `mixlen` of back pointers to originating zones (one-origin). Even if `mixlen` > 0, this argument is optional.
`mix_vf`	Array of length `mixlen` of volume fractions for the mixed zones.
`mixlen`	Length of the mixed data arrays (or zero if no mixed data is present). If `mixlen` > 0, then the "mix_" arguments describing the mixed data arrays must be non-null.
`datatype`	Volume fraction data type. One of the predefined SILO data types.

`optlist_id` Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.

`id` Returned status value. This is zero on success and -1 on failure. (The use of material identifiers is no longer supported.)

Returns:

dbputmat() returns zero on success and -1 on failure.

Description:

The dbputmat() function writes a material data object into the current open SILO file. The minimum required information for a material data object is supplied via the standard arguments to this function. The `optlist_id` argument must be used for supplying any information not requested through the standard arguments.

Notes:

The following table describes the options accepted by this function. See the section titled "Using the SILO Option Parameter" on page 3-2 for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_LABEL	character*(*)	Character string defining the label associated with material data.	DB_F77NULL
DBOPT_MAJORORDER	integer	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_TIME	real	Problem time value.	0.0

The model used for storing material data is the most efficient for MeshTV, and works as follows:

One zonal array, `matlist`, is used which contains the material number for a clean zone or an index into the mixed data arrays if the zone is mixed. Mixed zones are marked with negative entries in `matlist`, so you must take `ABS(matlist[i])` to get the actual 1-origin mixed data index. *All indices are 1-origin to allow* `matlist` *to use zero as a material number.*

The mixed data arrays are essentially a linked list of information about the mixed elements within a zone. Each mixed data array is of length `mixlen`. For a given index *i*, the following information is known about the *i*'th element:

`mix_zone[i]` The 1-origin index of the zone which contains this element.

`mix_mat[i]` The material number of this element

`mix_vf[i]` The volume fraction of this element

`mix_next[i]` The 1-origin index of the next material entry for this zone, else 0 if this is the last entry.

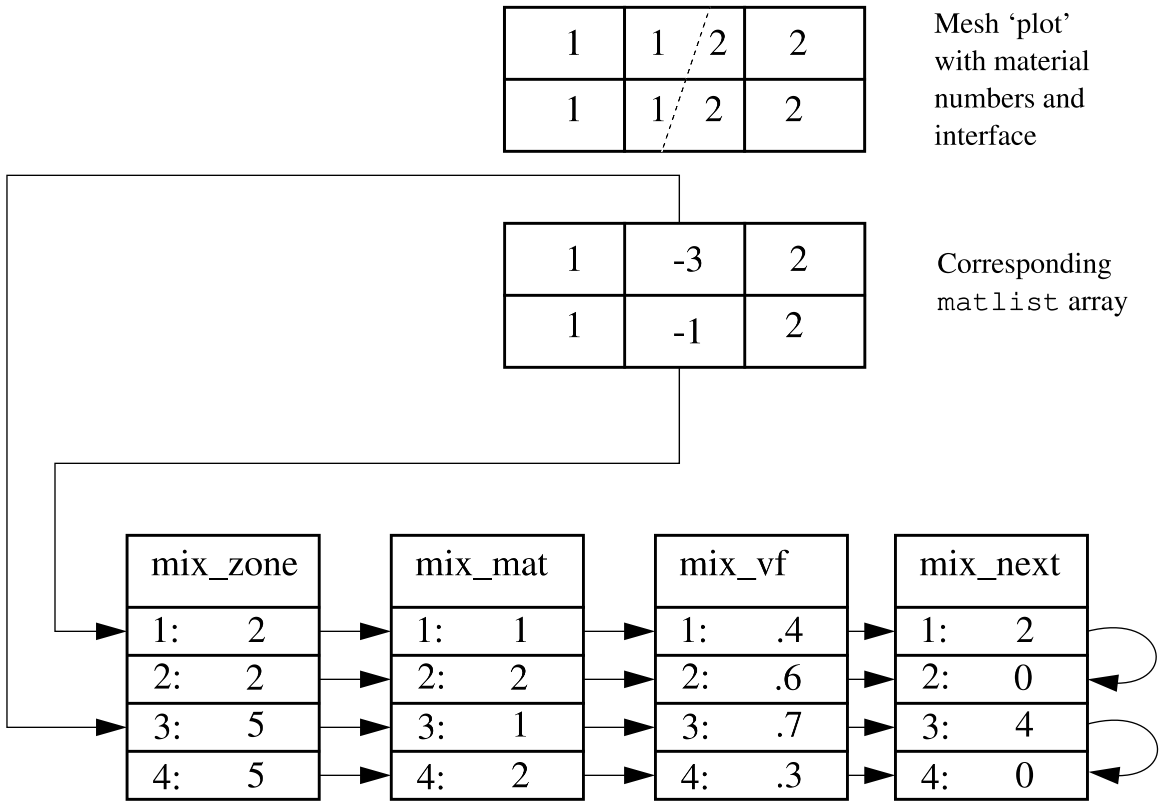


Figure 3-1: Example using mixed data arrays for representing material information.

`dbputmmat()`—Write a multi-block material object into the open SILO file

Synopsis:

```
include "silo.inc"
integer function dbputmmat (dbid, name, lname, nmat, matnames,
                  lmatnames, optlist_id, id)

integer dbid, lname, nmat, optlist_id, id
integer lmatnames(*)
character*(*) name
character*32 matnames(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the multi-material object.
`lname`	Length of the `name` parameter in characters.
`nmat`	Number of materials provided.
`matnames`	Array of length `nmat` containing the names of the materials to be associated with the multi-material object. This is an array of 32-character strings.
`lmatnames`	Array of length `nmat` containing the lengths of the `matnames` strings.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of multi-material identifiers is no longer supported.)

Returns:

dbputmmat() returns zero on success and -1 on failure.

Description:

The dbputmmat() function writes a multi-material object into the open SILO file.

dbputmmesh()—Write a multi-block mesh object into the open SILO file.

Synopsis:

```
include “silo.inc”
integer function dbputmmesh (dbid, name, lname, nmesh, meshnames,
                  lmeshnames, meshtypes, optlist_id, id)

integer dbid, lname, nmesh, lmeshnames(*)
integer meshtypes(*), optlist_id, id
character*32 meshnames(*)
character*(*) name
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the multi-block mesh structure.
`lname`	Length of the `name` parameter in characters.
`nmesh`	Number of meshes provided.
`meshnames`	Array of length `nmesh` containing the names of the meshes. This is an array of 32-character strings.
`lmeshnames`	Array of length `nmesh` containing the lengths of the `meshnames` strings.
`meshtypes`	Array of length `nmesh` containing the type of each mesh. One of the predefined types: DB_QUAD_RECT, DB_QUAD_CURV, DB_UCD.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of multimesh identifiers is no longer supported.)

Returns:

dbputmmesh() returns zero on success and -1 on failure.

Description:

The dbputmmesh() function writes a multi-block mesh object into the open SILO file. It accepts as input descriptions of the various sub-meshes (blocks) which are part of this mesh.

dbputmsp()—Write a material species data object into a SILO file

Synopsis:

```
include "silo.inc"
integer function dbputmsp (dbid, name, lname, matname, lmatname,
                  nmat, nmatspec, speclist, dims, ndims,
                  nspecies_mf, species_mf, mix_speclist, mixlen,
                  datatype, optlist_id, id)

integer dbid, lname, lmatname, nmat, ndims, nspecies_mf
integer mixlen, datatype, optlist_id, id
integer nmatspec(*), speclist(*), dims(*), mix_speclist(*)
character*(*) name, matname
real species_mf(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the material species data object.
`lname`	Length of the `name` parameter in characters.
`matname`	Name of the material object with which the material species object is associated.
`lmatname`	Length of the `matname` parameter in characters.
`nmat`	Number of materials.
`nmatspec`	Array of length `nmat` containing the number of material species associated with each material.
`speclist`	Array of dimension `ndims * dims` of indices into the species_mf array. Each entry corresponds to one zone. A positive value is the index to the mass fractions of a clean zone's material species. A negative value means that the zone is a mixed zone and that the array `mix_speclist` contains the index to the species mass fractions.
`dims`	Array of length `ndims` that defines the length of the `speclist` array.
`ndims`	Number of dimensions in the `speclist` array.
`nspecies_mf`	Number of material species mass fractions.
`species_mf`	Array of length `nspecies_mf` containing mass fractions of the material species.
`mix_speclist`	Array of length mixlen containing indices into the species_mf array. These are used for mixed zones.
`mixlen`	Length of the `mix_list` array.
`datatype`	The datatype of the mass fraction data in `species_mf`. One of the predefined SILO data types.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.

`id`	Returned status value. This is zero on success and -1 on failure. (The use of material species identifiers is no longer supported.)

Returns:

dbputmsp() returns zero on success and -1 on failure.

Description:

The dbputmsp() function writes a material species data object into the open SILO file. The minimum required information for a material species data object is supplied via the standard arguments to this function. The `optlist_id` argument must be used for supplying any information not requested through the standard arguments.

dbputmvar()—Write a multi-block variable object into the open SILO file.

Synopsis (Fortran):

```
include "silo.inc"
integer function dbputmvar (dbid, name, lname, nvar, varnames,
                   lvarnames, vartypes, optlist_id, id)

integer dbid, lname, nvar, optlist_id, id
integer lvarnames(*), vartypes(*)
character*(*) name
character*32 varnames(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the multi-block variable.
`lname`	Length of the `name` parameter in characters.
`nvar`	Number of variables associated with the multi-block variable.
`varnames`	Array of length `nvar` containing the names of the variables associated with each block. This is an array of 32-character strings.
`lvarnames`	Array of length `nvar` containing the lengths of the `varnames` strings.
`vartypes`	Array of length `nvar` containing the types of the variables. Each entry must be one of the following: DB_POINTVAR, DB_QUADVAR, or DB_UCDVAR.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of multi-block variable identifiers is no longer supported.)

Returns:

dbputmvar() returns zero on success and -1 on failure.

Description:

The dbputmvar() function writes a multi-block variable object into the open SILO file.

`dbputpm()`—Write a point mesh object into the open SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputpm (dbid, name, lname, ndims, x, y, z, nels,
                    datatype, optlist_id, id)

integer dbid, lname, ndims, nels, datatype, optlist_id, id
real x(*), y(*), z(*)
char*(*) name
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the mesh.
`lname`	Length of the `name` parameter in characters.
`ndims`	Number of dimensions.
`x, y, z`	Arrays containing coordinate values. If `ndims` is 2, then `z` is ignored.
`nels`	Number of elements (points) in the mesh.
`datatype`	Datatype of the coordinate arrays. One of the predefined SILO data types.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of pointmesh identifiers is no longer supported.)

Returns:

dbputpm() returns zero on success and -1 on failure.

Description:

The dbputpm() function accepts pointers to the coordinate arrays and is responsible for writing the mesh into a pointmesh object in the SILO file.

A SILO pointmesh object contains all necessary information for describing a mesh. This includes the coordinate arrays, the number of dimensions (1,2,3,...) and the number of points.

Notes:

The following table describes the options accepted by this function. See the section titled "Using the SILO Option Parameter" on page 3-2 for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_XLABEL	character*(*)	Character string defining the label associated with the X dimension.	DB_F77NULL

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_YLABEL	character*(*)	Character string defining the label associated with the Y dimension.	DB_F77NULL
DBOPT_ZLABEL	character*(*)	Character string defining the label associated with the Z dimension.	DB_F77NULL
DBOPT_NSPACE	integer	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	integer	Origin for arrays. Zero or one.	0
DBOPT_TIME	real	Problem time value.	0.0
DBOPT_XUNITS	character*(*)	Character string defining the units associated with the X dimension.	DB_F77NULL
DBOPT_YUNITS	character*(*)	Character string defining the units associated with the Y dimension.	DB_F77NULL
DBOPT_ZUNITS	character*(*)	Character string defining the units associated with the Z dimension.	DB_F77NULL

`dbputpv1()`—Write a scalar point variable object into the open SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputpv1 (dbid, name, lname, meshname, lmeshname,
                    var, nels, datatype, optlist_id, id)

integer dbid, lname, lmeshname, nels, datatype, optlist_id, id
character*(*) name, meshname
real var(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the variable.
`lname`	Length of the `name` parameter in characters
`meshname`	Name of the associated point mesh.
`lmeshname`	Length of the meshname parameter in characters.
`var`	Array containing data values for this variable.
`nels`	Number of elements (points) in variable.
`datatype`	Datatype of the variable. One of the predefined SILO data types.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of pointmesh variable objects is no longer supported.)

Returns:

dbputpv1() returns zero on success and -1 on failure.

Description:

The dbputpv1() function accepts a value array and is responsible for writing the variable into a point-variable object in the SILO file.

A SILO point-variable object contains all necessary information for describing a variable associated with a point mesh. This includes the number of arrays, the datatype of the variable, and the number of points. This function should be used when writing scalar quantities.

Notes:

The following table describes the options accepted by this function. See “Using the SILO Option Parameter” on page 3-2 for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_NSPACE	integer	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	integer	Origin for arrays. Zero or one.	0
DBOPT_TIME	real	Problem time value.	0.0

dbputqm()—Write a quad mesh object into the open SILO file.

Synopsis:

```
include "silo.inc
integer function dbputqm (dbid, name, lname, xname, lxname, yname,
                    lyname, zname, lzname, x, y, z, dims, ndims,
                    datatype, coordtype, optlist_id, id)

integer dbid, lname, lxname, lyname, lzname, ndims, datatype
integer coordtype, optlist_id, id
integer dims(*)
char*(*) name, xname, yname, zname
real x(*), y(*), z(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the mesh.
`lname`	Length of the `name` parameter in characters.
`[xyz]name`	Name to associate with the corresponding `[xyz]` coordinate array.
`l[xyz]name`	Length of the `[xyz]name` parameter in characters.
`x, y, z`	Arrays containing coordinate values. If `ndims` is 2, then `z` is ignored.
`dims`	Array of length `ndims` which describes the dimensionality of the mesh. Each value in the `dims` array indicates the number of nodes contained in the mesh along that dimension.
`ndims`	Number of dimensions.
`datatype`	Datatype of the coordinate arrays. One of the predefined SILO data types.
`coordtype`	Coordinate array type. One of the predefined types: DB_COLLINEAR or DB_NONCOLLINEAR. Collinear coordinate arrays are always one-dimensional, regardless of the dimensionality of the mesh; non-collinear arrays have the same dimensionality as the mesh.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of quadmesh identifiers is no longer supported.)

Returns:

dbputqm() returns zero on success and -1 on failure.

Description:

The dbputqm() function accepts pointers to the coordinate arrays and is responsible for writing the mesh into a quadmesh object in the SILO file.

A SILO quadmesh object contains all necessary information for describing a mesh. This includes the coordinate arrays, the rank of the mesh (1,2,3,...) and the type (collinear or non-collinear.) In addition, other information is useful and is therefore optionally included (row-major indicator, time and cycle of mesh, offsets to 'real' zones, plus coordinate system type.)

The following table describes the options accepted by this function. See the section titled "Using the SILO Option Parameter" on page 3-2 for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	integer	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_FACETYPE	integer	Zone face type. One of the pre-defined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_HI_OFFSET	integer *	Array of length `ndims` which defines zero-origin offsets from the last node for the ending index along each dimension.	{0,0,...}
DBOPT_LO_OFFSET	integer *	Array of length `ndims` which defines zero-origin offsets from the first node for the starting index along each dimension.	{0,0,...}
DBOPT_XLABEL	character*(*)	Character string defining the label associated with the X dimension.	DB_F77NULL
DBOPT_YLABEL	character*(*)	Character string defining the label associated with the Y dimension.	DB_F77NULL
DBOPT_ZLABEL	character*(*)	Character string defining the label associated with the Z dimension.	DB_F77NULL
DBOPT_MAJORORDER	integer	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_NSPACE	integer	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	integer	Origin for arrays. Zero or one.	0
DBOPT_PLANAR	integer	Planar value. One of: DB_AREA or DB_VOLUME.	DB_OTHER
DBOPT_TIME	real	Problem time value.	0.0
DBOPT_XUNITS	character*(*)	Character string defining the units associated with the X dimension.	DB_F77NULL

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_YUNITS	character*(*)	Character string defining the units associated with the Y dimension.	DB_F77NULL
DBOPT_ZUNITS	character*(*)	Character string defining the units associated with the Z dimension.	DB_F77NULL

The options DB_LO_OFFSET and DB_HI_OFFSET should be used if the mesh being described uses the notion of “phoney” zones (i.e., some zones should be ignored.) For example, if a 2-D mesh had designated the first column and row, and the last two columns and rows as “phoney”, then we would use: lo_off = {1,1} and hi_off = {2,2}.

dbputqv1()—Write a scalar quad variable object into the open SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputqv1 (dbid, name, lname, meshname, lmeshname,
                     var, dims, ndims, mixvar, mixlen, datatype,
                     centering, optlist_id, id)

integer dbid, lname, lmeshname, ndims, mixlen, datatype
integer centering, optlist_id, id
integer dims(*)
character*(*) name, meshname
real var(*), mixvar(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the variable.
`lname`	Length of the `name` parameter in characters.
`meshname`	Name of the mesh associated with this variable (written with dbputqm() or dbputum().) If no association is to be made, this value should be DB_F77NULL.
`lmeshname`	Length of the `meshname` parameter in characters.
`var`	Array defining the values associated with this variable.
`dims`	Array of length `ndims` which describes the dimensionality of the variable. Each value in the `dims` array indicates the number of elements contained in the variable along that dimension.
`ndims`	Number of dimensions.
`mixvar`	Array defining the mixed-data values associated with this variable. If no mixed values are present, this should be DB_F77NULL.
`mixlen`	Length of the mixed data arrays, if provided.
`datatype`	Datatype of sub-variables. One of the predefined SILO data types.
`centering`	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of quadmesh variable identifiers is no longer supported.)

Returns:

dbputqv1() returns zero on success and -1 on failure.

Description:

The dbputqv1() function writes a scalar variable associated with a quad mesh into a SILO file. Note that variables will be either node-centered or zone-centered. A quad-var object contains the variable values, plus the name of the associated quadmesh. Other information can also be included. This function should be used for writing scalar fields.

Notes:

The following table describes the options accepted by this function. See the section titled “Using the SILO Option Parameter” on page 3-2 for details on the use of this construct.

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	integer	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_FACETYPE	integer	Zone face type. One of the predefined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_LABEL	character*(*)	Character string defining the label associated with this variable.	DB_F77NULL
DBOPT_MAJORORDER	integer	Indicator for row-major (0) or column-major (1) storage for multidimensional arrays.	0
DBOPT_ORIGIN	integer	Origin for arrays. Zero or one.	0
DBOPT_TIME	real	Problem time value.	0.0
DBOPT_UNITS	character*(*)	Character string defining the units associated with this variable.	DB_F77NULL

dbputum()—Write an UCD mesh object into the open SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputum (dbid, name, lname, ndims, x, y, z,
                    xname, lxname, yname, lyname, zname, lzname,
                    datatype, nnodes, nzones, zlname, lzlname,
                    flname, lflname, optlist_id, id)

integer dbid, lname, lxname, lyname, lzname, datatype, nnodes
integer nzones, ndims, lzlname, lflname, optlist_id, id
real x(*), y(*), z(*)
character*(*) name, xname, yname, zname, zlname, flname
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the mesh.
`lname`	Length of the `name` parameter in characters.
`ndims`	Number of dimensions.
`x, y, z`	Arrays containing coordinate values. If `ndims` is 2, then `z` is ignored.
`[xyz]name`	Name to associate with the corresponding `[xyz]` coordinate array.
`l[xyz]name`	Length of the `[xyz]name` parameter in characters.
`datatype`	Datatype of the coordinate arrays. One of the predefined SILO data types.
`nnodes`	Number of nodes.
`nzones`	Number of zones.
`zlname`	Name of the zonelist structure associated with this variable (written with dbputzl().) If no association is to be made, this value should be DB_F77NULL.
`lzlname`	Length of the `zlname` parameter in characters.
`flname`	Name of the facelist structure associated with this variable (written with dbputfl().) If no association is to be made, this value should be DB_F77NULL.
`lflname`	Length of the `flname` parameter in characters.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of UCDmesh identifiers is no longer supported.)

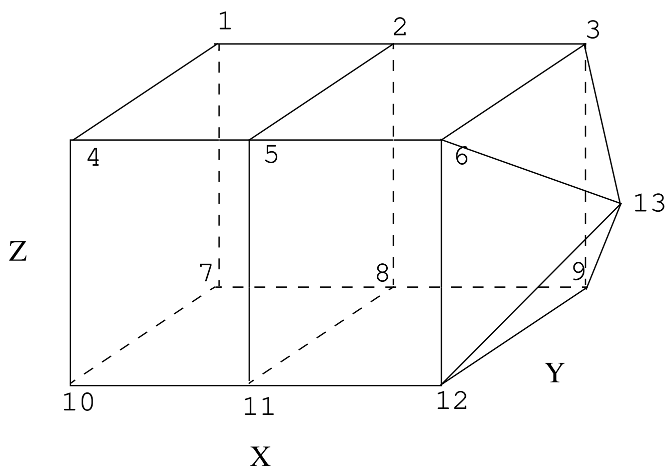
Returns:

dbputum() returns zero on success and -1 on failure.

Description:

The dbputum() function accepts pointers to the coordinate arrays and is responsible for writing the mesh into a UCD mesh object in the SILO file.

A SILO UCD mesh object contains all necessary information for describing a mesh. This includes the coordinate arrays, the rank of the mesh (1,2,3,...) and the type (collinear or non-collinear.) In addition, other information is useful and is therefore included (time and cycle of mesh, plus coordinate system type.)

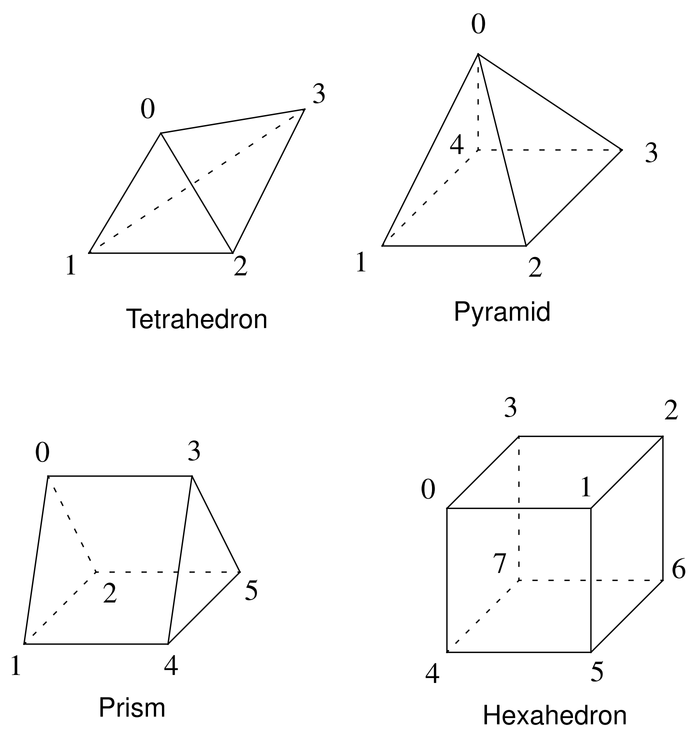



```
nnodes     = 13
nzones     = 3
nzshapes   = 2
lznodelist = 2*8 + 1*5 = 21 zone nodes
nfaces     = 13 external faces
nfshapes   = 2 external face shapes
nftypes    = 0
lfnodelist = 9*4 + 4*3 = 48 external face nodes

fnodelist = { 1,2,8,7 external face nodelist
              2,3,9,8,
              8,9,12,11,
              5,6,12,11,...}

fshapesize = {4,3} external face shape sizes
fshapecnt = {9,4} external face shape counts
fzoneno    = {1,2,2,2,...}external face zone nos

znodelist = { 1,4,5,2,7,10,11,8, zone nodelist
              2,5,6,3,8,11,12,9,
              13,3,6,12,9}
zshapesize = {8,5} zone shape sizes
zshapecnt = {2,1} zone shape counts
x = {0,1,2,0,1,2,0,1,2,0,1,2,3}
y = {1,1,1,0,0,0,1,1,1,0,0,0,.5}
z = {1,1,1,1,1,1,0,0,0,0,0,0,.5}
```

Figure 3-2: Example usage of UCD zonelist and *external* facelist variables.

Notes:

The order in which nodes are defined in the zonelist is important, especially for 3D cells. Nodes defining a 2D cell should be supplied in either clockwise or counterclockwise order around the cell. Nodes defining a 3D cell should be supplied in the order illustrated below:



Notes:Continued

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	integer	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_EDGELIST	integer	Object identifier for edgelist structure	0

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_FACETYPE	integer	Zone face type. One of the pre-defined types: DB_RECTILINEAR or DB_CURVILINEAR.	DB_RECTILINEAR
DBOPT_XLABEL	character*(*)	Character string defining the label associated with the X dimension.	DB_F77NULL
DBOPT_YLABEL	character*(*)	Character string defining the label associated with the Y dimension.	DB_F77NULL
DBOPT_ZLABEL	character*(*)	Character string defining the label associated with the Z dimension.	DB_F77NULL
DBOPT_NSPACE	integer	Number of spatial dimensions used by this mesh.	ndims
DBOPT_ORIGIN	integer	Origin for arrays. Zero or one.	0
DBOPT_PLANAR	integer	Planar value. One of: DB_AREA or DB_VOLUME.	DB_NONE
DBOPT_TIME	real	Problem time value.	0.0
DBOPT_XUNITS	character*(*)	Character string defining the units associated with the X dimension.	DB_F77NULL
DBOPT_YUNITS	character*(*)	Character string defining the units associated with the Y dimension.	DB_F77NULL
DBOPT_ZUNITS	character*(*)	Character string defining the units associated with the Z dimension.	DB_F77NULL

`dbputuv1()`—Write a scalar UCD variable object into the open SILO file.

Synopsis:

```
include "silo.inc"
integer function dbputuv1 (dbid, name, lname, meshname, lmeshname,
                   var, nels, mixvar, mixlen, datatype,
                   centering, optlist_id, id)

integer dbid, lname, lmeshname, nels, mixlen, datatype
integer centering, optlist_id, id
character*(*) name, meshname
real var(*), mixvar(*)
```

Arguments:

`dbid`	Database identifier.
`name`	Name of the variable.
`lname`	Length of the `name` parameter in characters.
`meshname`	Name of the mesh associated with this variable (written with dbputum().) If no association is to be made, this value should be DB_F77NULL.
`lmeshname`	Length of the `meshname` parameter in characters.
`var`	Array of length `nels` containing the values associated with this variable.
`nels`	Number of elements in this variable.
`mixvar`	Array of length `mixlen` containing the mixed-data values associated with this variable. If `mixlen` is zero, this value is ignored.
`mixlen`	Length of the `mixvar` array. If zero, no mixed data is present.
`datatype`	Datatype of variable. One of the predefined SILO data types.
`centering`	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
`optlist_id`	Option list identifier created with dbmkoptlist() and populated with dbaddopt(). If no options are to be provided, use DB_F77NULL for this argument.
`id`	Returned status value. This is zero on success and -1 on failure. (The use of UCDmesh variable identifiers is no longer supported.

Returns:

dbputuv1() returns zero on success and -1 on failure.

Description:

The dbputuv1() function writes a scalar variable associated with an UCD mesh into a SILO file. Note that variables will be either node-centered or zone-centered. A UCD variable object contains the variable values, plus the object identifier of the associated UCD mesh. Other information can also be included. This function is useful for writing scalar fields.

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_COORDSYS	integer	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_LABEL	character*(*)	Character strings defining the label associated with this variable.	DB_F77NULL
DBOPT_ORIGIN	integer	Origin for arrays. Zero or one.	0
DBOPT_TIME	real	Problem time value.	0.0
DBOPT_UNITS	character*(*)	Character string defining the units associated with this variable.	DB_F77NULL

dbputzl()—Write a zonelist object into a SILO file.

Synopsis:

```
include “silo.inc”
integer function dbputzl (dbid, name, lname, nzones, ndims,
                  nodelist, lnodelist, origin, shapesize,
                  shapecnt, nshapes, idzl)

integer dbid, lname, nzones, ndims, lnodelist, origin, nshapes,
                  idzl
integer nodelist(*), shapesize(*), shapecnt(*)
character*(*) name
```

Arguments:

`dbid`	Database identifier.
`name`	Name to assign this structure within the file.
`lname`	Length of the name parameter in characters.
`nzones`	Number of zones in associated mesh.
`ndims`	Number of spatial dimensions represented by associated mesh.
`nodelist`	Array of length `lnodelist` containing node indices describing mesh zones.
`lnodelist`	Length of the `nodelist` array.
`origin`	Origin for indices in the nodelist array. Should be zero or one.
`shapesize`	Array of length `nshapes` containing the number of nodes used by each zone shape.
`shapecnt`	Array of length `nshapes` containing the number of zones having each shape.
`nshapes`	Number of zone shapes.
`idzl`	Returned status value. This is zero on success and -1 on failure. (The use of zonelist identifiers is no longer supported.)

Returns:

dbputzl() returns zero on success and -1 on failure.

Description:

The dbputzl() function writes a zonelist object into the open SILO file. The name assigned to this object can in turn be used as the `zonel_name` parameter to the dbputum() function.

Notes:

See the write-up of dbputum() for a full description of the zonelist data structures.

`dbrdvar()`—Read a simple SILO variable.

Synopsis:

```
include "silo.inc"
integer function dbrdvar (dbid, varname, lvarname, result)

integer dbid, lvarname
character*(*) varname
real result(*)
```

Arguments:

`dbid`	Database identifier.
`varname`	Name of the variable
`lvarname`	Length of the `varname` parameter in characters.
`result`	Pointer to memory in which variable should be read. It is up to the application to provide sufficient space in which to read the variable.

Returns:

dbrdvar() returns zero on success and -1 on failure.

Description:

The dbrdvar() function reads a simple SILO variable into the given space.

dbrdvarslice()—Read a (hyper)slab of data from a SILO variable.

Synopsis:

```
include "silo.inc"
integer function dbrdvarslice (dbid, varname, lvarname, offset,
                    length, stride, ndims, result)

integer dbid, lvarname, ndims
integer offset(*), length(*), stride(*)
character*(*) varname
real result(*)
```

Arguments:

`dbid`	Database identifier.
`varname`	Name of the variable.
`lvarname`	Length of the `varname` parameter in characters.
`offset`	Array of length `ndims` of offsets in each dimension of the variable. This is 1-origin position from which to begin reading the slice.
`length`	Array of length `ndims` of lengths of data in each dimension to read from the variable. All lengths must be positive.
`stride`	Array of length `ndims` of stride steps in each dimension. If no striding is desired, zeroes should be passed in this array.
`ndims`	Number of dimensions in the variable.
`result`	Pointer to memory in which variable should be read. It is up to the application to provide sufficient space in which to read the variable.

Returns:

dbrdvarslice() returns zero on success and -1 on failure.

Description:

The dbrdvarslice() function reads a slab of data from a SILO variable into a location provided in the result array. Any hyperslab of data may be read.

Note that the minimum `length` value is 1 and the minimum `stride` value is zero.

A one-dimensional array slice:

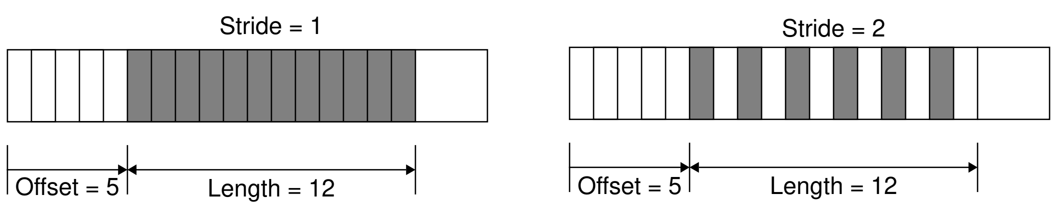


Figure 3-4: Array slice

`dbsetdir()`—Set the current directory within the SILO database.

Synopsis:

```
include "silo.inc"
integer function dbsetdir (dbid, pathname, lpathname)

integer dbid, lpathname
character*(*) pathname
```

Arguments:

`dbid`	Database identifier.
`pathname`	Pathname of the directory. This can be either an absolute or relative pathname.
`lpathname`	Length of the `pathname` string.

Returns:

dbsetdir() returns zero on success and -1 on failure.

Description:

The dbsetdir() function sets the current directory within the given SILO database.

`dbshowerrors()`—Set the error reporting mode.

Synopsis:

```
include "silo.inc"
integer function dbshowerrors (level)

integer level
```

Arguments:

`level`	Error reporting level. One of DB_ALL, DB_ABORT, DB_TOP, or DB_NONE.

Returns:

dbshowerrors() always returns zero. It cannot fail.

Description:

The dbshowerrors() function sets the level of error reporting done by SILO when it encounters an error. The following table describes the action taken upon error for different values of `level`:

Error level value	Error action
DB_ALL	Show all errors, beginning with the (possibly internal) routine that first detected the error and continuing up the call stack to the application.
DB_ABORT	Same as DB_ALL except abort() is called after the error message is printed.
DB_TOP	(Default) Only the top-level SILO functions issue error messages.
DB_NONE	The library does not handle error messages. The application is responsible for checking the return values of the SILO functions and handling the error.

For more information, see "Error Handling" on page 3-1.

`dbwrite()`—Write a simple SILO variable.

Synopsis:

```
#include “silo.inc”
integer function dbwrite (dbid, varname, lvarname, var, dims,
                   ndims, datatype)

integer dbid, lvarname, ndims, datatype
integer dims(*)
character*(*) varname
real var(*)
```

Arguments:

`dbid`	Database identifier.
`varname`	Name of the variable.
`lvarname`	Length of the `varname` parameter in characters.
`var`	Array defining the values associated with the variable.
`dims`	array of length `ndims` describing the dimensionality of the variable. Each value in the `dims` array indicates the number of elements contained in the variable along that dimension.
`ndims`	Number of dimensions.
`datatype`	Datatype of the variable. One of the predefined SILO data types.

Returns:

dbwrite() returns zero on success and -1 on failure.

Description:

The dbwrite() function writes a simple variable into a SILO file.

3.2. Summary of Fortran Interface

```
include "silo.inc"
integer function dbaddcopt (optlist_id, option, cvalue, lcvalue)
integer function dbadddopt (optlist_id, option, dvalue)
integer function dbaddiopt (optlist_id, option, ivalue)
integer function dbaddropt (optlist_id, option, rvalue)
integer function dbcalcfl (dbid, znodelist, nnodes, origin,
                           zshapesize, zshapecnt, nzshapes,
                           matlist, bnd_method, id)
integer function dbclose (dbid)
integer function dbcreate (pathname, lpathname, mode, target,
                           fileinfo, lfileinfo, filetype, dbid)
integer function dbfgetca (dbid, name, lname, values, nvalues)
integer function dbfreeoptlist (optlist_id)
integer function dbgetca (dbid, name, lname, enames, lenames,
                           elengths, nelems, values, nvalues,
                           datatype)
integer function dbinqca (dbid, name, lname, tlenames, nelems,
                           nvalues, datatype)
integer function dbinqlen (dbid, varname, lvarname, len)
integer function dbmkdir (dbid, dirname, ldirname, id)
integer function dbmkoptlist (maxopts, optlist_id)
integer function dbopen (name, lname, type, dbid)
integer function dbputca (dbid, name, lname, enames, lenames,
                           elengths, nelems, values, nvalues,
                           datatype, optlist_id)
integer function dbputfl (dbid, name, lname, nfaces, ndims,
                           nodelist, lnodelist, origin, zoneno,
                           shapesize, shapecnt, nshapes, types,
                           typelist, ntypes, idfl)
integer function dbputmat (dbid, name, lname, meshname, lmeshname,
                           nmat, matnos, matlist, dims, ndims,
                           mix_next, mix_mat, mix_zone, mix_vf,
                           mixlen, datatype, id)
integer function dbputmmat (dbid, name, lname, nmat, matnames,
                           lmatnames, optlist_id, id)
integer function dbputmmesh (dbid, name, lname, nmesh, meshids,
                           meshnames, lmeshnames, meshtypes,
                           meshdirs, optlist_id, id)
integer function dbputmsp (dbid, name, lname, matname, lmatname,
                           nmat, nmatspec, speclist, dims, ndims,
                           nspecies_mf, species_mf, mix_speclist,
                           mixlen, datatype, optlist_id, id)
integer function dbputmvar (dbid, name, lname, mmeshid, mmeshdir,
                           optlist_id, id)
integer function dbputpm (dbid, name, lname, ndims, x, y, z, nels,
                           datatype, optlist_id)
integer function dbputpv1 (dbid, name, lname, meshname, lmeshname,
                           var, nels, datatype, optlist_id, id)
```

```
integer function dbputqm (dbid, name, lname, xname, lxname, yname,
                          lyname, zname, lzname, x, y, z, dims,
                          ndims, datatype, coordtype,
                          optlist_id, id)
integer function dbputqv1 (dbid, name, lname, meshname, lmeshname,
                          var, dims, ndims, mixvar, mixlen,
                          datatype, centering, optlist_id, id)
integer function dbputum (dbid, name, lname, ndims, x, y, z,
                          xname, lxname, yname, lyname, zname,
                          lzname, datatype, nnodes, nzones,
                          zonelist_id, facelist_id, id)
integer function dbputuv1 (dbid, name, lname, meshname, lmeshname,
                          var, nels, mixvar, mixlen, datatype,
                          centering, id)
integer function dbputzl (dbid, name, lname, nzones, ndims,
                          nodelist, lnodelist, origin,
                          shapesize, shapecnt, nshapes, idzl)
integer function dbrdvar (dbid, varname, lvarname, result)
integer function dbrdvarslice (dbid, varname, lvarname, offset,
                          length, stride, ndims, result)
integer function dbsetdir (dbid, pathname, lpathname)
integer function dbshowerrors (level)
integer function dbwrite (dbid, varname, lvarname, var, dims,
                          ndims, datatype)
```

Appendix A

Constant Definitions and Data Structures

A.1 Constant Definitions

Table A-1. Zone Face Type Values and Definitions

Zone-Face Type	Definition
DB_RECTILINEAR	100
DB_CURVILINEAR	101

Table A-2. Centering Values and Definitions

Centering	Definition
DB_NODECENT	110
DB_ZONECENT	111
DB_FACECENT	112
DB_NOTCENT	115

Table A-3. Coordinate System Values and Definitions

Coordinate System	Definition
DB_CARTESIAN	120
DB_CYLINDRICAL	121
DB_SPHERICAL	122

Table A-3. Coordinate System Values and Definitions

Coordinate System	Definition
DB_NUMERICAL	123
DB_OTHER	124

Table A-4. Coordinate Array Values and Definitions

Coordinate Type	Definition
DB_COLLINEAR	130
DB_NONCOLLINEAR	131

Table A-5. Planar Values and Definitions

Planar Type	Definition
DB_AREA	140
DB_VOLUME	141

Table A-6. Major Order Values and Definitions

Major Order	Definition
DB_ROWMAJOR	0
DB_COLMAJOR	1

Table A-7. Datatype Values and Definitions

Datatype	Definition
DB_INT	SILO_INT
DB_SHORT	SILO_SHORT
DB_LONG	SILO_LONG
DB_FLOAT	SILO_FLOAT
DB_DOUBLE	SILO_DOUBLE
DB_CHAR	SILO_CHAR

Table A-8. Options and Associated Datatypes

Option Name	Value Datatype
DBOPT_ALIGN	float
DBOPT_COORDSYS	int
DBOPT_CYCLE	int
DBOPT_FACETYPE	int
DBOPT_HI_OFFSET	int *
DBOPT_LO_OFFSET	int *
DBOPT_LABELS	char *
DBOPT_XLABELS	char *
DBOPT_YLABELS	char *
DBOPT_ZLABELS	char *
DBOPT_MAJORORDER	int
DBOPT_NSPACE	int
DBOPT_ORIGIN	int
DBOPT_PLANAR	int
DBOPT_TIME	float
DBOPT_UNITS	char *
DBOPT_XUNITS	char *
DBOPT_YUNITS	char *
DBOPT_ZUNITS	char *

A.2 C Data Structures

A.2.1 Compound Array Definition

```
typedef struct {
        int id;                 /* identifier of the compound array */
        char *name;             /* name of the compound array */
        char **elemnames;       /* names of the simple array elements */
        int *elemlengths;       /* lengths of the simple arrays */
        int nelems;             /* number of simple arrays */
        void *values;           /* simple array values */
        int nvalues;            /* sum reduction of `elemlengths' vector */
        int datatype;           /* simple array element data type */
} DBcompoundarray;
```

A.2.2 Curve Definition

```
typedef struct {
    /*----------- X vs. Y (Curve) Data -----------*/
    int id;                  /* Identifier for this object */
    int datatype;            /* Datatype for x and y (float, double) */
    int origin;              /* ‘0’ or ‘1’ */
    char *title;             /* Title for curve */
    char *xvarname;          /* Name of domain (x) variable */
    char *xlabel;            /* Label for x-axis */
    char *ylabel;            /* Label for y-axis */
    char *xunits;            /* Units for domain */
    char *yunits;            /* Units for range */
    float *x;                /* Domain values for curve */
    float *y;                /* Range values for curve */
    int npts;                /* Number of points in curve */
} DBcurve;
```

A.2.3 Material Data Definition

```
typedef struct {
    /*----------- Material Information -----------*/
    int id;                  /* Identifier */
    char *name;              /* Name of this material information data */
    int ndims;               /* Rank of ‘matlist’ variable */
    int origin;              /* ‘0’ or ‘1’ */
    int dims[3];             /* Number of elements in each dimension */
    int major_order;         /* ‘1’ indicates row-major */
    int stride[3];           /* Offsets to adjacent elements in matlist*/
    int nmat;                /* Number of materials */
    int *matnos;             /* Array [nmat] of valid material numbers */
    int *matlist;            /* Array [nzone] with mat. number or mix index */
    int mixlen;              /* Length of mixed data arrays (mix_xxx) */
    int datatype;            /* Datatype of volume-fractions (double,float) */
    float *mix_vf;           /* Array [mixlen] of volume fractions */
    int *mix_next;           /* Array [mixlen] of mixed data indices */
    int *mix_mat;            /* Array [mixlen] of material numbers */
    int *mix_zone;           /* Array [mixlen] of back pointers to mesh */
} DBmaterial;
```

A.2.4 Material Species Data Definition

```
typedef struct {
    /*-----------  Species Information -----------*/
    int id;                  /* Identifier */
    char *name;              /* Name of this material species information block */
    char *matname;           /* Name of material object with which the material
                                species object is associated */
    int nmat;                /* Number of materials */
    int *nmatspec;           /* Array [nmat] of the number of material species
```

```
                                    associated with each material */
    int ndims;                    /* Rank of 'speclist' variable */
    int dims[3];                  /* Number of elements in each dimension of the 'spe-
                                    clist' variable */
    int major_order;              /* '1' indicates row-major for multi-dimensional
                                    arrays */
    int stride[3];                /* Offsets to adjacent elements in 'speclist' */
    int origin;                   /* '0' or '1' */
    int nspecies_mf;              /* Total number of species mass fractions */
    float *species_mf;            /* Array [nspecies_mf] of mass fractions of material
                                    species */
    int *speclist;                /* Zone array of dimensions described by ndims and
                                    dims.  Each element of the array is an index into one
                                    of the species mass fraction arrays.  A positive value
                                    is the index in the 'species_mf' array of the mass
                                    fractions of the clean zone's material species:
                                    species_mf[speclist[i]] is the mass fraction of the
                                    first species of material matlist[i] in zone i.  A nega-
                                    tive value means that the zone is a mixed zone and
                                    that the array mix_speclist contains the index to the
                                    species mass fracitons;  -speclist[i] is the index in the
                                    'mix_speclist' array for zone i */
    int mixlen;                   /* Length of 'mix_speclist' array */
    int *mix_speclist;            /* Array [mixlen] of 1-origin indices into the
                                    'species_mf' array.  species_mf[mix_speclist[j]] is
                                    the index in array 'species_mf' of the first of the
                                    mass fractions for material mix_mat[j] */
    int datatype;                 /* Datatype of mass fraction data */
} DBmatspecies;
```

A.2.5 Mesh Variable Definition

```
typedef struct {
    /*----------- Generic Mesh-Data Variable -----------*/
    int id;                         /* Identifier for this object */
    char *name;                   /* Name of variable */
    char *units;                  /* Units for variable, e.g, 'mm/ms' */
    char *label;                  /* Label (perhaps for editing purposes) */
    int cycle;                    /* Problem cycle number */
    float time;                   /* Problem time */
    double dtime;                 /* Problem time, double data type */
    int meshid;                   /* Identifier for associated mesh */
    float **vals;                 /* Array of pointers to data arrays */
    int datatype;                 /* Type of data pointed to by 'val' */
    int nels;                     /* Number of elements in each array */
    int nvals;                    /* Number of arrays pointed to by 'vals' */
    int nspace;                   /* Spatial rank of variable */
    int ndims;                    /* Rank of 'vals' array(s) (computational rank) */
    int origin;                   /* '0' or '1' */
    int centering;                /* Centering within mesh (nodal, zonal, other) */
```

```
    float align[3];          /* Alignment per dimension if centering==other*/
    int dims[3];             /* Number of elements in each dimension */
    int major_order;         /* 1 indicates row-major for multi-dimensional arrays
                                */
    int stride[3];           /* Offsets to adjacent elements  */
    int min_index[3];        /* Index in each dimension of 1st non-phoney */
    int max_index[3];        /* Index in each dimension of last non-phoney*/
} DBmeshvar;
```

A.2.6 Multi-Block Mesh Definition

```
typedef struct {
    /*----------- Multi-Block Mesh -----------*/
    int id;                  /* Identifier for this object */
    int nblocks;             /* Number of blocks in mesh */
    int *meshids;            /* Array of mesh-id's which comprise this mesh */
    char **meshnames;        /* Array of mesh-names corresponding to meshids */
    int *meshtypes;          /* Array of mesh-type indicators [nblocks] */
    int *dirids;             /* Array of directory ID's which contain block */
} DBmultimesh;
```

A.2.7 Multi-Block Variable Definition

```
typedef struct {
    /*----------- Multi-Block Variable -----------*/
    int id;                  /* Identifier for this object */
    int meshid;              /* Object-id of associated multi-block mesh */
    int meshdir;             /* Directory ID of dir containing multi-block mesh */
} DBmultivar;
```

A.2.8 Optlist Definition

```
typedef struct {
    /* Option structure for some of the C SILO functions */
    int *options;            /* Array of option identifiers (see Table above) */
    void **values;           /* Array of pointers to option values*/
    int numopts;             /* Number of options in use */
    int maxopts;             /* Maximum number of options (i.e., length of arrays)
                                */
} DBoptlist;
```

A.2.9 Point Mesh Definition

```
typedef struct {
    /*----------- Point Mesh-----------*/
    int id;                  /* Identifier for this object */
    int block_no;            /* Block number for this mesh */
    char *name;              /* Name associated with mesh */
    int cycle;               /* Problem cycle number */
    float time;              /* Problem time */
```

```
    char *units[3];           /* Units for variable, e.g, 'mm/ms' */
    char *labels[3];          /* Label associated with each dimension */
    char *title;              /* Title for curve */
    float *coords[3];         /* Mesh node coordinates */
    int datatype;             /* Datatype of coordinate arrays (double,float) */
    int ndims;                /* Number of computational dimensions */
    int nels;                 /* Total number of elements (points) in mesh*/
    int origin;               /* '0' or '1' */
} DBpointmesh;
```

A.2.10 Quad Mesh Definition

```
typedef struct {
    /*----------- Quad Mesh -----------*/
    int id;                   /* Identifier for this object */
    int block_no;             /* Block number for this mesh */
    char *name;               /* Name associated with mesh */
    int cycle;                /* Problem cycle number */
    float time;               /* Problem time */
    int coord_sys;            /* Cartesian, cylindrical, spherical */
    int major_order;          /* '1' indicates row-major for multi-d arrays */
    int stride[3];            /* Offsets to adjacent elements */
    int coordtype;            /* Coord array type: collinear, non-collinear */
    int facetype;             /* Zone face type: rect, curv */
    int planar;               /* Sentinel: zones represent area or volume? */
    float *coords[3];         /* Mesh node coordinate ptrs [ndims] */
    int datatype;             /* Datatype of coordinate arrays (double,float) */
    float min_extents[3];     /* Min mesh extents [ndims] */
    float max_extents[3];     /* Max mesh extents [ndims] */
    char *labels[3];          /* Label associated with each dimension */
    char *units[3];           /* Units for variable, e.g, 'mm/ms' */
    int ndims;                /* Number of computational dimensions */
    int nspace;               /* Number of physical dimensions */
    int nnodes;               /* Total number of nodes */
    int dims[3];              /* Number of nodes per dimension */
    int origin;               /* '0' or '1' */
    int min_index[3];         /* Index in each dimension of first non-phoney */
    int max_index[3];         /* Index in each dimension of last non-phoney */
} DBquadmesh;
```

A.2.11 Quad Variable Definition

```
typedef struct {
    /*----------- Quad Variable -----------*/
    int id;                   /* Identifier for this object */
    char *name;               /* Name of variable */
    char *units;              /* Units for variable, e.g, 'mm/ms' */
    char *label;              /* Label (perhaps for editing purposes) */
    int cycle;                /* Problem cycle number */
    float time;               /* Problem time */
```

```
    int meshid;             /* Identifier for associated mesh */
    float **vals;           /* Array of pointers to data arrays */
    int datatype;           /* Type of data pointed to by 'val' */
    int nels;               /* Number of elements in each array */
    int nvals;              /* Number of arrays pointed to by 'vals' */
    int ndims;              /* Rank of variable */
    int dims[3];            /* Number of elements in each dimension */
    int major_order;        /* '1' indicates row-major for multi-d arrays */
    int stride[3];          /* Offsets to adjacent elements */
    int min_index[3];       /* Index in each dimension of first non-phoney */
    int max_index[3];       /* Index in each dimension of last non-phoney */
    int origin;             /* '0' or '1' */
    float align[3];         /* Centering and alignment per dimension */
    float **mixvals;        /* nvals pointers to data arrays for mixed zones */
    int mixlen;             /* Number of elements in each mixed zone data array
                               */
    int use_specmf;         /* Flag indicating whether to apply species mass frac-
                               tions to the variable. */
} DBquadvar;
```

A.2.12 Table of Contents Definiton

```
typedef struct {
    char  **curve_names      /* Array [ncurve] of pointers to curve names */
    int   *curve_ids         /* Array [ncurve] of curve identifiers */
    int   ncurve             /* Number of curves */
    char  **multimesh_names  /* Array [nmultimesh] of pointers to multimesh names
                                */
    int   *multimesh_ids     /* Array [nmultimesh] of multimesh identifiers */
    int   nmultimesh         /* Number of multimeshes */
    char  **multivar_names   /* Array [nmultivar] of pointers to multi-variable
                                names */
    int   *multivar_ids      /* Array [nmultivar] of multi-variable identifiers */
    int   nmultivar          /* Number of multi-variables */
    char  **qmesh_names      /* Array [nqmesh] of pointers to quadmesh names */
    int   *qmesh_ids         /* Array [nqmesh] of quadmesh identifiers */
    int   nqmesh             /* Number of quadmeshes */
    char  **qvar_names       /* Array [nqvar] of pointers to quadmesh variable
                                names */
    int   *qvar_ids          /* Array [nqvar] of quadmesh variable identifiers */
    int   nqvar              /* Number of quadmesh variables */
    char  **ucdmesh_names    /* Array [nucdmesh] of pointers to ucdmesh names */
    int   *ucdmesh_ids       /* Array [nucdmesh] of ucdmesh identifiers */
    int   nucdmesh           /* Number of ucdmeshes */
    char  **ucdvar_names     /* Array [nucdvar] of pointers to ucdmesh variable
                                names */
    int   *ucdvar_ids        /* Array [nucdvar] of ucdmesh variable identifiers */
    int   nucdvar            /* Number of ucdmesh variables */
    char  **ptmesh_names     /* Array [nptmesh] of pointers to pointmesh names */
    int   *ptmesh_ids        /* Array [nptmesh] of pointmesh identifiers */
```

```
    int   nptmesh             /* Number of pointmeshes */
    char  **ptvar_names       /* Array [nptvar] of pointers to pointmesh variable
                                 names */
    int   *ptvar_ids          /* Array [nptvar] of pointmesh variable identifiers */
    int   nptvar              /* Number of pointmesh variables */
    char  **mat_names         /* Array [nmat] of pointers to materialnames */
    int   *mat_ids            /* Array [nmat] of material identifiers */
    int   nmat                /* Number of materials */
    char  **matspecies_names  /* Array [nmatspecies] of pointers to material species
                                 names */
    int   *matspecies_ids     /* Array [nmatspecies] of material species identifiers
                                 */
    int   nmatspecies         /* Number of material species */
    char  **var_names         /* Array [nvar] of pointers to variable names */
    int   *var_ids            /* Array [nvar] of variable identifiers */
    int   nvar                /* Number of variables */
    char  **obj_names         /* Array [nobj] of poitners to object names */
    int   *obj_ids            /* Array [nobj] of object identifiers */
    int   nobj                /* Number of objects */
    char  **dir_names         /* Array [ndir] of pointers to subdirectory names */
    int   *dir_ids            /* Array [ndir] of subdirectory identifiers */
    int   ndir                /* Number of subdirectories */
    char  **array_names       /* Array [narrays] of pointers to array names */
    int   *array_ids          /* Array [narrays] of array identifiers */
    int   narrays             /* Number of arrays */
} DBtoc;
```

A.2.13 UCD Mesh Definition

```
typedef struct {
    /*----------- Unstructured Cell Data (UCD) Mesh -----------*/
    int id;                   /* Identifier for this object */
    int block_no;             /* Block number for this mesh */
    char *name;               /* Name associated with mesh */
    int cycle;                /* Problem cycle number */
    float time;               /* Problem time */
    int coord_sys;            /* Coordinate system */
    char *units[3];           /* Units for variable, e.g, 'mm/ms' */
    char *labels[3];          /* Label associated with each dimension */
    float *coords[3];         /* Mesh node coordinates */
    int datatype;             /* Datatype of coordinate arrays (double,float) */
    float min_extents[3];     /* Min mesh extents [ndims] */
    float max_extents[3];     /* Max mesh extents [ndims] */
    int ndims;                /* Number of computational dimensions */
    int nnodes;               /* Total number of nodes */
    int origin;               /* '0' or '1' */
    DBfacelist *faces;        /* Data structure describing mesh faces */
    DBzonelist *zones;        /* Data structure describing mesh zones */
    DBedgelist *edges;        /* Data structure describing mesh edges (optional) */
} DBucdmesh;
```

A.2.14 UCD Variable Definition

```
typedef struct {
    /*----------- Unstructured Cell Data (UCD) Variable -----------*/
    int id;                  /* Identifier for this object */
    char *name;              /* Name of variable */
    int cycle;               /* Problem cycle number */
    char *units;             /* Units for variable, e.g, 'mm/ms' */
    char *label;             /* Label (perhaps for editing purposes) */
    float time;              /* Problem time */
    int meshid;              /* Identifier for associated mesh */
    float **vals;            /* Array of pointers to data arrays */
    int datatype;            /* Type of data pointed to by 'vals' */
    int nels;                /* Number of elements in each array */
    int nvals;               /* Number of arrays pointed to by 'vals' */
    int ndims;               /* Rank of variable */
    int origin;              /* '0' or '1' */
    int centering;           /* Centering within mesh (nodal or zonal) */
} DBucdvar;
```

A.2.15 UCD Edgelist Definition

```
typedef struct {
    int ndims;               /* Number of dimensions (2,3) */
    int nedges;              /* Number of edges */
    int *edge_beg;           /* [nnodes] */
    int *edge_end;           /* [nedges] */
    int origin;              /* '0' or '1' */
} DBedgelist;
```

A.2.16 UCD Facelist Definition

```
typedef struct {
    /*----------- Required components ------------*/
    int ndims;               /* Number of dimensions (2,3) */
    int nfaces;              /* Number of faces in list */
    int origin;              /* '0' or '1' */
    int *nodelist;           /* Sequential list of nodes which comprise faces */
    int lnodelist;           /* Number of nodes in nodelist */
                               /*----------- 3D components ------------------*/
    int nshapes;             /* Number of face shapes */
    int *shapecnt;           /* [nshapes] Number of occurences of each shape */
    int *shapesize;          /* [nshapes] Number of nodes per shape */
                               /*----------- Optional type component---------*/
    int ntypes;              /* Number of face types */
    int *typelist;           /* [ntypes] Type ID for each type */
    int *types;              /* [nfaces] Type info for each face */
                               /*----------- Optional zone-reference component-------
                               --*/
    int *zoneno;             /* [nfaces] Zone number for each face */
```

```
} DBfacelist;
```

A.2.17 UCD Zonelist Definition

```
typedef struct {
    int ndims;              /* Number of dimensions (2,3) */
    int nzones;             /* Number of zones in list */
    int nshapes;            /* Number of zone shapes */
    int *shapecnt;          /* [nshapes] Number of occurences of each shape */
    int *shapesize;         /* [nshapes] Number of nodes per shape */
    int *nodelist;          /* Sequential list of nodes which comprise zones */
    int lnodelist;          /* Number of nodes in nodelist */
    int origin;             /* '0' or '1' */
} DBzonelist;
```

Glossary

Block	Also known as a mesh-block. This is the fundamental building block of a computational mesh. It defines the nodal coordinates of one contiguous section of a mesh.
Curve	X versus Y data. This object contains the domain and range values, along with the number of points in the curve. In addition, a title, variable names, labels, and units may be provided.
Facelist	Face-oriented connectivity information for a UCD mesh. This object contains a sequential list of nodes which identifies the faces in the mesh, and arrays which describe the shape(s) of the faces in the mesh. It may optionally include arrays which provide type information for each face.
Material	A physical material being modeled in a computer simulation. This includes the number of materials present, a list of valid material identifiers, and a zonal-length array which contains the material identifiers for each zone.
Material species	Extra material information. A material species is a type of a material. They are used when a given material (i.e. air) may be made up of other materials (i.e. oxygen, nitrogen) in differing amounts.
Mesh	A computational mesh, composed of one or more mesh-blocks. A mesh can be composed of mesh-blocks of different types (quad, UCD) as well as of different shapes.
Multimat	A set of materials. This object contains the names of the materials in the set.
Multimesh	A set of meshes. This object contains the names of and types of the meshes in the set.
Multivar	Mesh variable data associated with a multimesh.

Node A mathematical point. The fundamental building-block of a mesh or zone.

Pointmesh A point mesh. This includes dimension and coordinate data.

Quadmesh A quadrilateral mesh. This includes the dimension and coordinate data, but typically also includes the mesh's coordinate system, labelling and unit information, minimum and maximum extents, and valid index ranges.

Quadvar A variable associated with a quadrilateral mesh. This includes the variable's data, centering information (node-centered vs. zone centered), and the name of the quad mesh with which this variable is associated. Additional information, such as time, cycle, units, label, and index ranges can also be included.

UCD Unstructured cell data is a term commonly used to denote an arbitrarily connected mesh. Such a mesh is composed of vectors of coordinate values along with an index array which identifies the nodes associated with each zone and/or face. Zones may contain any number of nodes for 2-D meshes, and either four, five, six, or eight nodes for 3-D meshes.

Ucdmesh An unstructured mesh. This includes the dimension, connectivity, and coordinate data, but typically also includes the mesh's coordinate system, labelling and unit information, minimum and maximum extents, and a list of face indices.

Ucdvar A variable associated with a UCD mesh. This includes the variable's data, centering information (node-centered vs. zone-centered), and the name of the UCD mesh with which this variable is associated. Additional information, such as time, cycle, units, and label can also be included.

Variable Array data. This object contains, in addition to the data, the dimensions and data type of the array. This object is not required to be (but usually is) associated with a mesh.

Zone An area or volume from which meshes are comprised. Zones are polygons or polyhedra with nodes as vertices.

Zonelist Zone-oriented connectivity information for a UCD mesh. This object contains a sequential list of nodes which identifies the zones in the mesh, and arrays which describe the shape(s) of the zones in the mesh.