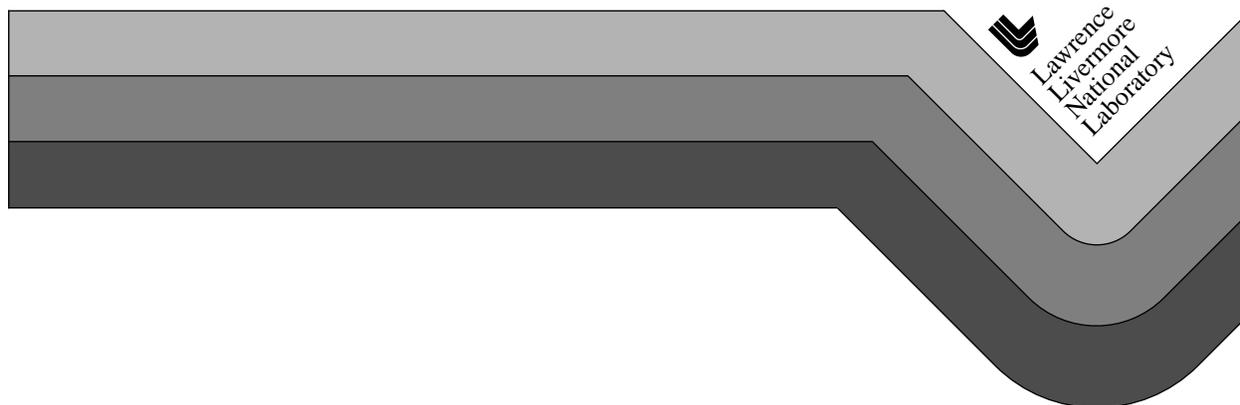


# *Silo User's Guide*

*Revision: March 2002*

*Version: 4.3*



## DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

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

```
int DBAddDblComponent(Object *object, char
 *compname, double d);
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 *DBCAllocExternalFacelist(
 int nodelist[], int nnodes,
 int origin, int shapsize[],
 int shapecnt[], int nshapes,
 int matlist[], int bnd_method);
DBfacelist *DBCAllocExternalFacelist2(
 int nodelist[], int nnodes,
 int lo_offset, int hi_offset,
 int origin, int shapetype[],
 int shapsize[], int shapecnt[],
 int nshapes, 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 *DBErrMsgString(void);
void *DBFortranAccessPointer(int value);
int DBFortranAllocPointer(void *);
void DBFortranRemovePointer(int);
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);
int DBGetComponentType(DBfile *dbfile, char
 *objname, char *compname);
DBcompoundarray *DBGetComponentarray(
 DBfile *dbfile, char *arrayname);
DBcurve *DBGetCurve(DBfile *dbfile,
 char *curvename);
long DBGetDataReadMask(void);
int DBGetDir(DBfile *dbfile, char *dirname);
DBmaterial *DBGetMaterial(DBfile *dbfile,
 char *mat_name);
DBmatspecies *DBGetMatspecies(DBfile *dbfile,
 char *ms_name);
DBmultimat *DBGetMultimat(DBfile *dbfile,
 char *name);
DBmultimesh *DBGetMultimesh(DBfile *dbfile,
 char *meshname);
DBmultivar *DBGetMultivar(DBfile *dbfile,
 char *varname);
DBmultimatspecies *DBGetMultimatspecies(
 DBfile *dbfile, char *name);
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 DBGetVarType(DBfile *dbfile,
 char *varname);
int DBInqCompoundarray(DBfile *dbfile, char*
 name, char *elemnames[], int *elemlengths,
 int nelems, int nvalues, int datatype);
int DBInqFile(char *filename);
int DBInqMeshname(DBfile *dbfile,
 char *varname, char *meshname);
int DBInqMeshtype(DBfile *dbfile,
 char *meshname);

int DBInqVarExists(DBfile *dbfile,
 char *name);
DBObjectType DBInqVarType(DBfile *dbfile,
 char *name);
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 mode);
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 DBPutCurve(DBfile *dbfile,
 char *curvename, void *xvals,
 void *yvals, int datatype, int npoints,
 DBOptlist *optlist);
int DBPutFacelist(DBfile *dbfile, char *name,
 int nfaces, int ndims, int nodelist[],
 int lnodelist, int origin, int zoneno[],
 int shapsize[], 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 DBPutMultimatspecies(DBfile *dbfile,
 char *name, int nspec, char *specnames[],
 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 shapetype[], int shapecnt[],
int nshapes);
int DBPutZonelist2(DBfile *dbfile,
char *name, int nzones, int ndims,
int nodelist[], int lnodelist, int origin,
int lo_offset, int hi_offset,
int shapetype[], int shapetype[],
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);
long DBGetDataReadMask(long mask);
int DBSetDir(DBfile *dbfile, char *pathname);
void DBShowErrors(int level,
void(*func)(char*));
char *DBVersion(void);

```

```

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);
int DBWriteSlice(DBfile *dbfile, char *varname,
void *var, int datatype, int *offset, int
*len, int *stride, int *dims, int ndims);

```

## Fortran Functions

```

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, zshape, 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 dbgetcurve(dbid, curvename,
lcurvename, maxpoints, xvals, yvals,
datatype, npoints)
integer function dbinqca(dbid, name, lname,
tlenames, nelems, nvalues, datatype)
integer function dbinqfile(filename)
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,
mode, dbid)
integer function dbputca(dbid, name, lname,
enames, lenames, elengths, nelems, values,
nvalues, datatype, optlist_id, id)
integer function dbputcurve (dbid, curvename,
lcurvename, xvals, yvals, datatype,
npoints, optlist_id, id)
integer function dbputfl(dbid, name, lname,
nfaces, ndims, nodelist, lnodelist, origin,
zoneno, shapetype, 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 dbputpvl(dbid, name, lname,
meshname, lmeshname, var, nels, datatype,
optlist_id, id)
integer function dbputqm(dbid, name, lname,
xname, lxname, yname, lymname, zname,
lzname, x, y, z, dims, ndims, datatype,
coordtype, optlist_id, id)
integer function dbputqvl(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 dbputuvl(dbid, name, lname,
meshname, lmeshname, var, nels, mixvar,
mixlen, datatype, centering, id)
integer function dbputz1(dbid, name, lname,
nzones, ndims, nodelist, lnodelist,
origin, shapetype, shapecnt, nshapes,
idz1)
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)
integer function dbwriteslice(dbid, varname,
lvarname, var, datatype, offset, length,
stride, ndims)

```

---

# Table of Contents

---

<b>List of Figures</b> .....	vii
------------------------------	-----

## **Chapter 1. Introduction to Silo**

Overview .....	1-1
Silo Architecture .....	1-2
Reading Silo Files .....	1-2
Writing Silo files .....	1-2
Terminology .....	1-2
Computational Meshes Supported by Silo .....	1-3
Quadrilateral-Based Meshes and Related Data .....	1-3
UCD-Based Meshes and Related Data .....	1-4
Point Meshes and Related Data .....	1-5
Silo Objects .....	1-5
Silo Directories .....	1-7

## **Chapter 2. C Functions**

C Interface Overview .....	2-1
Error Handling .....	2-1
Optional Arguments .....	2-2
Using the Silo Option Parameter .....	2-2
C Calling Sequence .....	2-2

---

## **Chapter 3. Fortran Functions**

Fortran Interface.....	3-1
Error Handling .....	3-1
Optional Arguments.....	3-1
Using the Silo Option Parameter .....	3-2
Fortran Calling Sequence.....	3-2
<b>Appendix A. Data Structures.....</b>	<b>A-1</b>
<b>Glossary.....</b>	<b>G-1</b>

---

# List of Figures

---

	Figure 1-1: Model of Silo Architecture. ....	1-2
	Figure 1-2: Examples of quadrilateral meshes. ....	1-3
	Figure 1-3: Phoney zones around a collinear quadrilateral mesh. ....	1-4
	Figure 1-4: Sample 2-D UCD Meshes.....	1-4
	Figure 1-5: UCD 2-D and 3-D Cell Shapes.....	1-5
	Figure 1-6: Sample hierarchy within a Silo file.....	1-7
	Figure 2-1: Example using mixed data arrays for representing material information.....	2-77
	Figure 2-2: Example usage of UCD zonelist and external facelist variables. ....	2-99
	Figure 2-3: Node ordering for UCD zone shapes. ....	2-100
	Figure 2-4: Example usage of UCD zonelist combining a hex and 2 polyhedra.....	2-101
	Figure 2-5: Array slice.....	2-113
	Figure 2-6: Array slice.....	2-123
	Figure 3-1: Example using mixed data arrays for representing material information.....	3-28
	Figure 3-2: Node ordering for UCD zone shapes. ....	3-47
	Figure 3-3: Example usage of UCD zonelist and external facelist variables. ....	3-48
	Figure 3-4: Array slice.....	3-54
	Figure 3-5: Array slice.....	3-59

---

---

# Chapter 1 Introduction to Silo

---

## 1.1. Overview

Silo is a library which implements an application programming 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)<sup>1</sup> 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 PDBext (PDBLib extensions) interface to read objects written by that interface(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.

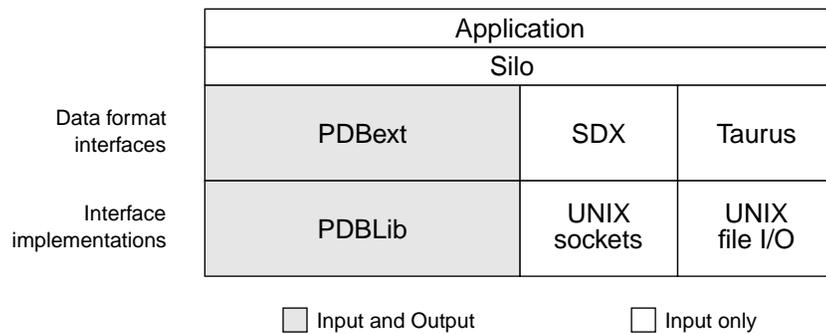


Figure 1-1: Model of Silo Architecture.

## 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).

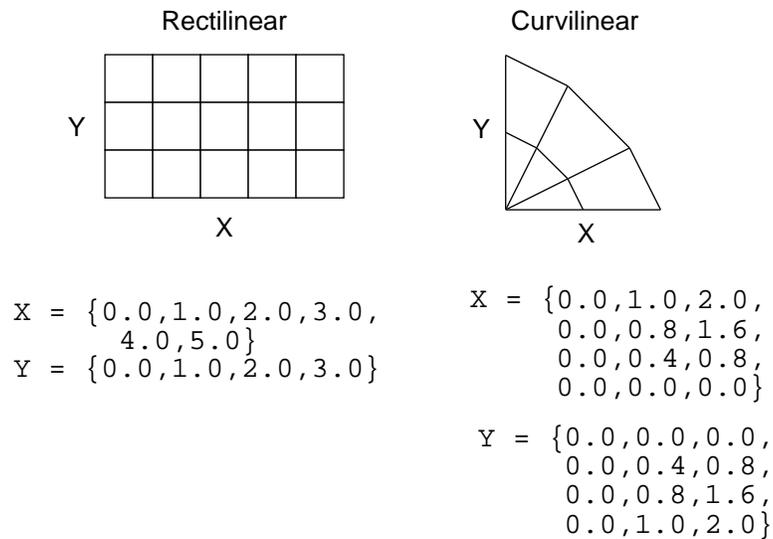


Figure 1-2: Examples of quadrilateral meshes.

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.

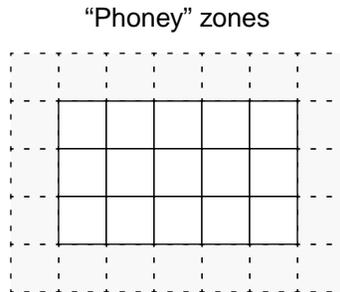


Figure 1-3: Phoney zones around a collinear quadrilateral mesh.

#### 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*.

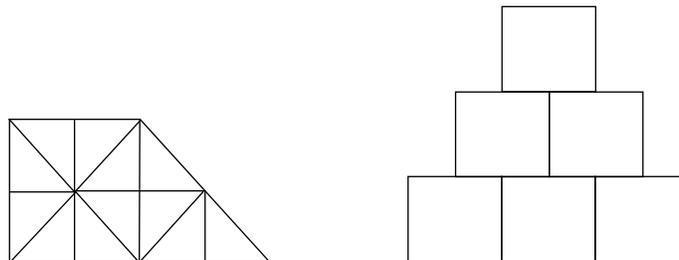


Figure 1-4: Sample 2-D UCD Meshes

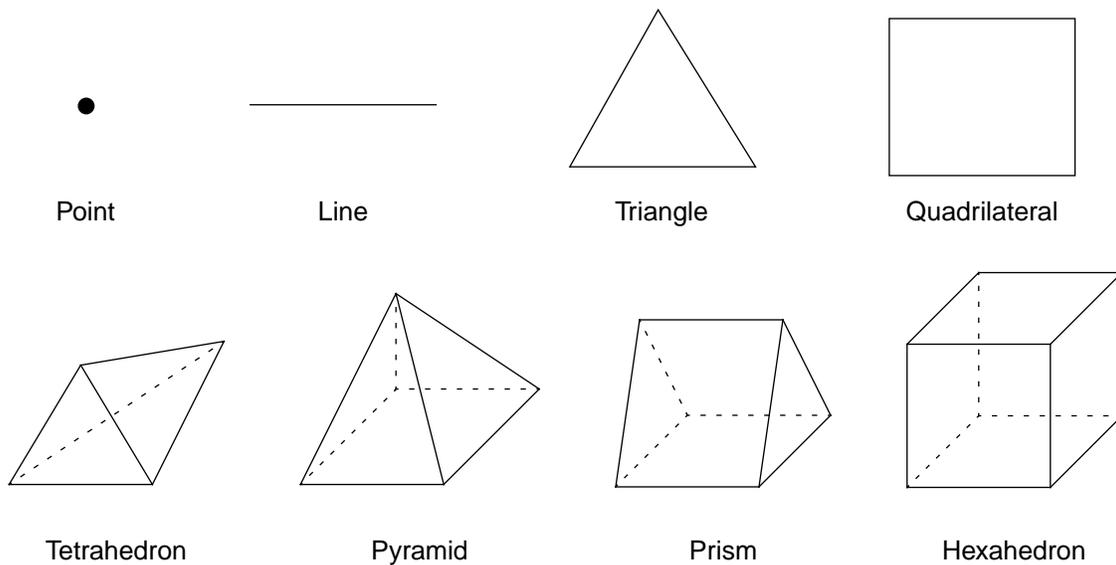


Figure 1-5: UCD 2-D and 3-D Cell Shapes

#### 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 <sup>1</sup> . 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.
Multimatspecies	A set of material species. This object contains the names of the material species 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.

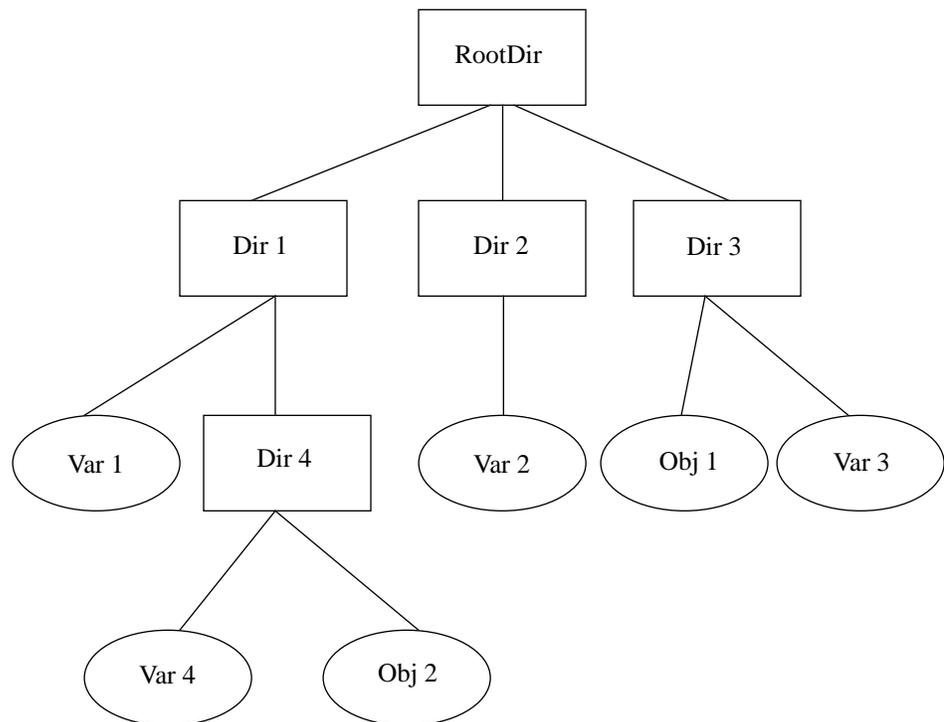


Figure 1-6: Sample hierarchy within a Silo file.



## 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 <code>abort()</code> 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 handling 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 `siloh.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 (
    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 <siloh.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 */
DBCclose(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
    . . .

DBCclose
```

#### 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 <silos.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, DB_READ);

/* 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("facettype: %d\n", qm->facettype);
    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 */
DBCclose(file);

return (0);
}
```

Table 2-1. C Interface Functions and Fortran Equivalents

C Functions	Fortran Equivalent	C Functions	Fortran Equivalent
DBAddDbIComponent	—	DBGetVarLength	—
DBAddFltComponent	—	DBInqCompoundarray	dbinqca
DBAddIntComponent	—	DBInqFile	dbinqfile
DBAddOption	dbaddopt...	DBInqMeshname	—
DBAddStrComponent	—	DBInqMeshtype	—
DBAddVarComponent	—	DBInqVarExists	—
DBAlloc...	—	DBInqVarType	—
DBCalcExternalFacelist	dbcacfl	DBMakeObject	—
DBCalcExternalFacelist2	—	DBMakeOptlist	dbmkoptlist
DBClearObject	—	DBMkDir	dbmkdir
DBClearOptlist	—	DBOpen	dbopen
DBCclose	dbcclose	DBPause	—
DBContinue	—	DBPutCompoundarray	dbputca
DBCreate	dbcreate	DBPutCurve	dbputcurve
DBErrFunc	—	DBPutFacelist	dbputfl
DBErrno	—	DBPutMaterial	dbputmat
DBErrString	—	DBPutMatspecies	dbputmsp
DBFortranAccessPointer	—	DBPutMultimat	dbputmmat
DBFortranAllocPointer	—	DBPutMultimatspecies	—
DBFortranRemovePointer	—	DBPutMultimesh	dbputmmesh
DBFreeObject	—	DBPutPointmesh	dbputpm
DBFreeOptlist	dbfreeoptlist	DBPutPointvar	—
DBFree...	—	DBPutMultivar	dbputmvar
DBGetAtt	—	DBPutPointvar1	dbputpv1
DBGetComponent	—	DBPutQuadmesh	dbputqm
DBGetComponentType	—	DBPutQuadvar	—
DBGetCompoundarray	dbgetca	DBPutQuadvar1	dbputqv1
DBGetCurve	dbgetcurve	DBPutUcdmesh	dbputum
DBGetDataReadMask	—	DBPutUcdvar	—
DBGetDir	—	DBPutUcdvar1	dbputuv1
DBGetMaterial	—	DBPutZonelist	dbputzl
DBGetMatspecies	—	DBPutZonelist2	—

**Table 2-1. C Interface Functions and Fortran Equivalents**

C Functions	Fortran Equivalent	C Functions	Fortran Equivalent
DBGetMultimat	—	DBReadAtt	—
DBGetMultimatspecies	—	DBReadVar	dbrdvar
DBGetMultimesh	—	DBReadVar1	—
DBGetMultivar	—	DBReadVarSlice	dbrdvarslice
DBGetPointmesh	—	DBSetDataReadMask	—
DBGetPointvar	—	DBSetDir	dbsetdir
DBGetQuadmesh	—	DBShowErrors	dbshowerrors
DBGetQuadvar	—	DBVersion	—
DBGetToc	—	DBWrite	dbwrite
DBGetUcdmesh	—	DBWriteComponent	—
DBGetUcdvar	—	DBWriteObject	—
DBGetVar	—	DBWriteSlice	dbwriteslice
DBGetVarByteLength	—		

**DBAddDblComponent**—Add a double precision floating point component to an object.

*Synopsis:*

```
int DBAddDblComponent (DObject *object, char *compname, double d)
```

*Arguments:*

object	Pointer to an object. This object is created with the DBMakeObject function.
compname	The component name.
d	The value of the double precision floating point component.

*Returns:*

DBAddDblComponent returns zero on success and -1 on failure.

*Description:*

The DBAddDblComponent function adds a component of double precision floating point data to an existing object.

**DBAddFltComponent**—Add a floating point component to an object.

*Synopsis:*

```
int DBAddFltComponent (DBObject *object, char *compname, double f)
```

*Arguments:*

<code>object</code>	Pointer to an object. This object is created with the <code>DBMakeObject</code> function.
<code>compname</code>	The component name.
<code>f</code>	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:*

```
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:*

```
int DBAddOption (DBoptlist *optlist, int option, void *value)
```

*Arguments:*

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

*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:*

```
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:*

```
int DBAddVarComponent (DBoject *object, char* compname,  
                      char *vardata)
```

*Arguments:*

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

*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:*

```
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 1 in Appendix A for a description of these data structures.

---

**DBCalcExternalFacelist**—Calculate an external facelist for a UCD mesh.

*Synopsis:*

```
DBfacelist *DBCalcExternalFacelist (int nodelist[], int nnodes,
                                     int origin, int shapessize[],
                                     int shapecnt[], int nshapes, int matlist[],
                                     int bnd_method)
```

*Arguments:*

nodelist	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.
shapessize	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.
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 zones. Faces get generated for both zones sharing a common face. This setting should not be used with meshes that contain mixed material zones. If this setting is used with meshes with mixed material zones, all faces which border a mixed material zone will be include. The matlist parameter must be provided.

For a description of how to nodes for the allowed shares are enumerated, see “DBPutUcdmesh” on page 2-98.

**DBCcalcExternalFacelist2**—Calculate an external facelist for a UCD mesh containing ghost zones.

*Synopsis:*

```
DBfacelist *DBCcalcExternalFacelist2 (int nodelist[], int nnodes,
                                     int low_offset, int hi_offset, int origin,
                                     int shapetype[], int shapysize[],
                                     int shapecnt[], int nshapes, int matlist[],
                                     int bnd_method)
```

*Arguments:*

- nodelist      Array of node indices describing mesh zones.
- nnodes        Number of nodes in associated mesh.
- lo\_offset     The number of ghost zones at the beginning of the nodelist.
- hi\_offset     The number of ghost zones at the end of the nodelist.
- origin        Origin for indices in the nodelist array. Should be zero or one.
- shapetype     Array of length nshapes containing the type of each zone shape. See description below.
- shapysize     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.
- matlist       Array containing material numbers for each zone (else NULL).
- bnd\_method    Method to use for calculating external faces. See description below.

*Returns:*

DBCcalcExternalFacelist2 returns a DBfacelist pointer on success and NULL on failure.

*Description:*

The DBCcalcExternalFacelist2 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 zones. Faces get generated for both zones sharing a common face. This setting should not be used with meshes that contain mixed material zones. If this setting is used with meshes with mixed material zones, all faces which border a mixed material zone will be included. The matlist parameter must be provided.

The allowed shape types are described in the following table:

Type	Description
DB_ZONETYPE_BEAM	A line segment
DB_ZONETYPE_POLYGON	A polygon where nodes are enumerated to form a polygon
DB_ZONETYPE_TRIANGLE	A triangle
DB_ZONETYPE_QUAD	A quadrilateral
DB_ZONETYPE_POLYHEDRON	A polyhedron with nodes enumerated to form faces and faces are enumerated to form a polyhedron
DB_ZONETYPE_TET	A tetrahedron
DB_ZONETYPE_PYRAMID	A pyramid
DB_ZONETYPE_PRISM	A prism
DB_ZONETYPE_HEX	A hexahedron

For a description of how the nodes for the allowed shapes are enumerated, see “DBPutUcdmesh” on page 2-98.

**DBCclearObject**—Clear an object.

*Synopsis:*

```
int DBCclearObject (DObject *object)
```

*Arguments:*

object	Pointer to the object to be cleared. This object is created with the DBMakeObject function.
--------	---

*Returns:*

DBCclearObject returns zero on success and -1 on failure.

*Description:*

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

**DBCclearOptlist**—Clear an optlist.

*Synopsis:*

```
int DBCclearOptlist (DBoptlist *optlist)
```

*Arguments:*

<code>optlist</code>	Pointer to an option list structure containing option/value pairs. This structure is created with the <code>DBMakeOptlist</code> function.
----------------------	--

*Returns:*

DBCclearOptlist returns zero on success and -1 on failure.

*Description:*

The DBCclearOptlist function removes all options from the given option list.

**DBCclose**—Close a Silo database.

*Synopsis:*

```
int DBClose (DBfile *dbfile)
```

*Arguments:*

dbfile            Database file pointer.

*Returns:*

DBCclose returns zero on success and -1 on failure.

*Description:*

The DBCclose function closes a Silo database.

**DBContinue**—Continues a simulation

*Synopsis:*

```
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:*

```
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:*

```
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:*

```
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:*

```
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.

**DBFortranAccessPointer**—Access Silo objects created through the Fortran Silo interface.

*Synopsis:*

```
void *DBFortranAccessPointer (int value)
```

*Arguments:*

value            The value returned by a Silo Fortran function, referencing a Silo object.

*Returns:*

DBFortranAccessPointer returns a pointer to a Silo object (which must be cast to the appropriate type) on success, and NULL on failure.

*Description:*

The DBFortranAccessPointer function allows programs written in both C and Fortran to access the same data structures. Many of the routines in the Fortran interface to Silo return an “object id”, an integer which refers to a Silo object. DBFortranAccessPointer converts this integer into a C pointer so that the sections of code written in C can access the Silo object directly.

See “DBFortranAllocPointer” on page 2-28 and “DBFortranRemovePointer” on page 2-29 for more information about how to use Silo objects in code that uses C and Fortran together.

## **DBFortranAllocPointer**—Facilitates accessing Silo objects through Fortran

### *Synopsis:*

```
int DBFortranAllocPointer (void *pointer)
```

### *Arguments:*

`pointer`      A pointer to a Silo object for which a Fortran identifier is needed

### *Returns:*

DBFortranAllocPointer returns an integer that Fortran code can use to reference the given Silo object.

### *Description:*

The DBFortranAllocPointer function allows programs written in both C and Fortran to access the same data structures. Many of the routines in the Fortran interface to Silo use an “object id”, an integer which refers to a Silo object. DBFortanAllocPointer converts a pointer to a Silo object into an integer that Fortran code can use. In some ways, this function is the inverse of DBFortranAccessPointer.

The integer that DBFortranAllocPointer returns is used to index a table of Silo object pointers. When done with the integer, the entry in the table may be freed for use later through the use of DBFortranRemovePointer.

See “DBFortranAccessPointer” on page 2-27 and “DBFortranRemovePointer” on page 2-29 for more information about how to use Silo objects in code that uses C and Fortran together.

**DBFortranRemovePointer**—Removes a pointer from the Fortran-C index table

*Synopsis:*

```
void DBFortranRemovePointer (int value)
```

*Arguments:*

value            An integer returned by DBFortranAllocPointer

*Returns:*

Nothing

*Description:*

The DBFortranRemovePointer function frees up the storage associated with Silo object pointers as allocated by DBFortranAllocPointer.

Code that uses both C and Fortran may make use of DBFortranAllocPointer to allocate space in a translation table so that the same Silo object may be referenced by both languages. DBFortranAccessPointer provides access to this Silo object from the C side. Once the Fortran side of the code is done referencing the object, the space in the translation table may be freed by calling DBFortranRemovePointer.

See “DBFortranAccessPointer” on page 2-27 and “DBFortranAllocPointer” on page 2-28 for more information about how to use Silo objects in code that uses C and Fortran together.

**DBFree...**—Release memory associated with a Silo structure.

*Synopsis:*

```
void DBFreeCompoundarray (DBcompoundarray *x)
void DBFreeEdgelist (DBedgelist *x)
void DBFreeFacelist (DBfacelist *x)
void DBFreeMaterial (DBmaterial *x)
void DBFreeMatspecies (DBmatspecies *x)
void DBFreeMeshvar (DBmeshvar *x)
void DBFreeMultimesh (DBmultimesh *x)
void DBFreeMultivar (DBmultivar *x)
void DBFreePointmesh (DBpointmesh *x)
void DBFreeQuadmesh (DBquadmesh *x)
void DBFreeQuadvar (DBquadvar *x)
void DBFreeUcdmesh (DBucdmesh *x)
void DBFreeUcdvar (DBucdvar *x)
void 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:*

```
int DBFreeObject (DObject *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:*

```
int DBFreeOptlist (DBoptlist *optlist)
```

*Arguments:*

<code>optlist</code>	Pointer to an option list structure containing option/value pairs. This structure is created with the <code>DBMakeOptlist</code> 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:*

```
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:*

```
void *DBGetComponent (DBfile *dbfile, char *objname,  
                 char *compname)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>objname</code>	Object name.
<code>compname</code>	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.

**DBGetComponentType**—Return the type of an object component.

*Synopsis:*

```
int DBGetComponentType (DBfile *dbfile, char *objname,  
                   char *compname)
```

*Arguments:*

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

*Returns:*

The values that are returned depend on the component's type and how the component was written into the object. The component types and their corresponding return values are listed in the table below.

Component Type	Return value
Integer	DB_INT
Float	DB_FLOAT
Double	DB_DOUBLE
String	DB_CHAR
Variable	DB_VARIABLE
<i>all others</i>	DB_NOTYPE

*Description:*

The DBGetComponentType function reads the component's type and returns it. If either the object or component does not exist, DB\_NOTYPE is returned. This function allows the application to process the component without having to know its type in advance.

**DBGetCompoundarray**—Read a compound array from a Silo database.

*Synopsis:*

```
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 1 in Appendix A for the definition of DBcompoundarray.

**DBGetCurve**—Read a curve from a Silo database.

*Synopsis:*

```
DBcurve *DBGetCurve (DBfile *dbfile, char *curvename)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>curvename</code>	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 1 in Appendix A for the definition of DBcurve.

**DBGetDataReadMask**—Get the current data read mask*Synopsis:*

```
long DBGetDataReadMask (void)
```

*Returns:*

DBGetDataReadMask returns the current data read mask.

*Description:*

The DBGetDataReadMask allows the user to find out what mask is currently being used to read the data within Silo objects.

Most Silo objects have a metadata portion and a data portion. The data portion is that part of the object that consists of pointers to long arrays of data. These arrays are “problem sized”.

Setting the data read mask (using the DBSetDataReadMask call) allows for a DBGet\* call to only return part of the data. With the data read mask set to DBAll, the DBGet\* functions return all of the information. With the data read mask set to DBNone, they return only the metadata. The mask is a bit vector specifying which part of the data model should be read.

A special case is found in the DBCalc flag. Sometimes data is not stored in the file, but is instead calculated from other information. The DBCalc flag controls this behavior. If it is turned off, the data is not calculated. If it is turned on, the data is calculated.

The values that DBGetDataReadMask returns are binary-or’ed combinations of the values shown in the following table:

Mask bit	Meaning
DBAll	All data values are read. This value is identical to specifying all of the other mask bits or’ed together, setting all of the bit values to 1.
DBNone	No data values are read. This value sets all of the bit values to 0.
DBCalc	If data is calculable, calculate it. Otherwise, return NULL for that information.
DBMatMatnos	The lists of material numbers in material objects are read by the DBGetMaterial call.
DBMatMatnames	The array of material names in material objects are read by the DBGetMaterial call.
DBMatMatlist	The lists of the correspondence between zones and material numbers in material objects are read by the DBGetMaterial call.
DBMatMixList	The lists of mixed material information in material objects are read by the DBGetMaterial call.
DBCurveArrays	The data values of curves are read by the DBGetCurve call.
DBPMCcoords	The coordinate values of pointmeshes are read by the DBGetPointmesh call.

Mask bit	Meaning
DBPVData	The data values of pointvars are read by the DBGetPointvar call.
DBQMCoords	The coordinate values of quadmeshes are read by the DBGetQuadmesh call.
DBQVData	The data values of quadvars are read by the DBGetQuadvar call.
DBUMCoords	The coordinate values of UCD meshes are read by the DBGetUcdmesh call.
DBUMFacelist	The facelists of UCD meshes are read by the DBGetUcdmesh call.
DBUMZonelist	The zonelists of UCD meshes are read by the DBGetUcdmesh call.
DBUVDData	The data values of UCD variables are read by the DBGetUcdvar call.
DBFacelistInfo	The nodelists and shape information in facelists are read by the DBGetFacelist call.
DBZonelistInfo	The nodelist and shape information in zonelists are read by the DBGetZonelist call.

Note: The data read mask is currently recognized only by the following drivers: PDB, Taurus.

**DBGetDir**—Get the name of the current directory.

*Synopsis:*

```
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:*

```
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 2 in Appendix A for the definition of DBmaterial.

**DBGetMatspecies**—Read material species data from a Silo database.

*Synopsis:*

```
DBmatspecies *DBGetMatspecies (DBfile *dbfile, char *ms_name)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>ms_name</code>	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 2 in Appendix A for the definition of DBmatspecies.

**DBGetMultimat**—Read a multi-block material object from a Silo database

*Synopsis:*

```
DBmultimat *DBGetMultimat (DBfile *dbfile, char *name)
```

*Arguments:*

dbfile	Database file pointer
name	Name of the multi-block material object

*Returns:*

DBGetMultimat returns a pointer to a DBmultimat structure on success and NULL on failure.

*Description:*

The DBGetMultimat function allocates a DBmultimat data structure, reads a multi-block material from the Silo database, and returns a pointer to that structure. If an error occurs, NULL is returned.

*Notes:*

See “Multi-Block Material Definition” on page 4 in Appendix A for the definition of DBmultimat.

**DBGetMultimatspecies**—Read a multi-block species from a Silo database.

*Synopsis:*

```
DBmultimesh *DBGetMultimatspecies (DBfile *dbfile, char *name)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the multi-block material species.

*Returns:*

DBGetMultimatspecies returns a pointer to a DBmultimatspecies structure on success and NULL on failure.

*Description:*

The DBGetMultimatspecies function allocates a DBmultimatspecies data structure, reads a multi-block material species from the Silo database, and returns a pointer to that structure. If an error occurs, NULL is returned.

*Notes:*

See “Multi-Block Species Definition” on page 4 in Appendix A for the definition of DBmultimatspecies.

**DBGetMultimesh**—Read a multi-block mesh from a Silo database.

*Synopsis:*

```
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 3 in Appendix A for the definition of DBmultimesh.

**DBGetMultivar**—Read a multi-block variable definition from a Silo database.

*Synopsis:*

```
DBmultivar *DBGetMultivar (DBfile *dbfile, char *varname)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>varname</code>	Name of the multi-block variable.

*Returns:*

DBGetMultivar returns a pointer to a DBmultivar structure on success and NULL on failure.

*Description:*

The DBGetMultivar function allocates a DBmultivar data structure, reads a multi-block variable from the Silo database, and returns a pointer to that structure. If an error occurs, NULL is returned.

*Notes:*

See “Multi-Block Variable Definition” on page 4 in Appendix A for the definition of DBmultivar.

**DBGetPointmesh**—Read a point mesh from a Silo database.

*Synopsis:*

```
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 5 in Appendix A for the definition of DBpointmesh.

**DBGetPointvar**—Read a point variable from a Silo database.

*Synopsis:*

```
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 “Point Variable Definition” on page 3 in Appendix A for the definition of DBmeshvar.

**DBGetQuadmesh**—Read a quadrilateral mesh from a Silo database.

*Synopsis:*

```
DBquadmesh *DBGetQuadmesh (DBfile *dbfile, char *meshname)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>meshname</code>	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 5 in Appendix A for the definition of DBquadmesh.

**DBGetQuadvar**—Read a quadrilateral variable from a Silo database.

*Synopsis:*

```
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 6 in Appendix A for the definition of DBquadvar.

**DBGetToc**—Get the table of contents of a Silo database.

*Synopsis:*

```
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 7 in Appendix A for the definition of DBtoc.

**DBGetUcdmesh**—Read a UCD mesh from a Silo database.

*Synopsis:*

```
DBucdmesh *DBGetUcdmesh (DBfile *dbfile, char *meshname)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>meshname</code>	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 7 in Appendix A for the definition of DBucdmesh.

**DBGetUcdvar**—Read a UCD variable from a Silo database.

*Synopsis:*

```
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 8 in Appendix A for the definition of DBucdvar.

**DBGetVar**—Allocate space for, and return, a simple variable.

*Synopsis:*

```
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 simple 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 simple variable.

*Synopsis:*

```
int DBGetVarByteLength (DBfile *dbfile, char *varname)
```

*Arguments:*

dbfile	Database file pointer.
varname	Variable name.

*Returns:*

DBGetVarByteLength returns the length of the given simple variable in bytes on success and -1 on failure.

*Description:*

The DBGetVarByteLength function returns the length of the requested simple variable, in bytes. This is useful for determining how much memory to allocate before reading a simple 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 simple variable.

*Synopsis:*

```
int DBGetVarLength (DBfile *dbfile, char *varname)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>varname</code>	Variable name.

*Returns:*

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

*Description:*

The DBGetVarLength function returns the length of the requested simple variable, in number of elements. For example a 16 byte array containing 4 floats has 4 elements.

**DBGetVarType**—Return the Silo datatype of a simple variable.

*Synopsis:*

```
int DBGetVarType (DBfile *dbfile, char *varname)
```

*Arguments:*

dbfile	Database file pointer.
varname	Variable name.

*Returns:*

DBGetVarType returns the Silo datatype of the given simple variable on success and -1 on failure.

*Description:*

The DBGetVarType function returns the Silo datatype of the requested simple variable. For example, DB\_FLOAT for float variables.

*Notes:*

This only works for simple Silo variables (those written using DBWrite or DBWriteSlice). To query the type of other variables, use DBInqVarType instead.

## **DBInqCompoundarray**—Inquire Compound Array attributes.

### *Synopsis:*

```
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.

**DBInqFile**—Inquire if `filename` is a Silo file.

*Synopsis:*

```
int DBInqFile (char *filename)
```

*Arguments:*

`filename`      Name of file.

*Returns:*

DBInqFilename returns 0 if `filename` is not a Silo file, a positive number if `filename` is a Silo file, and a negative number if an error occurred.

*Description:*

The DBInqFile function is mainly used for its return value, as seen above.

**DBInqMeshname**—Inquire the mesh name associated with a variable.

*Synopsis:*

```
int DBInqMeshname (DBfile *dbfile, char *varname, char *meshname)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>varname</code>	Variable name.
<code>meshname</code>	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:*

```
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

## **DBInqVarExists**—Queries variable existence

*Synopsis:*

```
int DBInqVarExists (DBfile *dbfile, char *name);
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Object name.

*Returns:*

**I** DBInqVarExists returns non-zero if the object exists in the file. Zero otherwise.

*Description:*

The DBInqVarExists function is used to check for existence of an object in the given file.

If an object was written to a file, but the file has yet to be DBClose'd, the results of this function querying that variable are undefined.

---

**DBInqVarType**—Return the type of the given object

*Synopsis:*

```
DBObjectType DBInqVarType (DBfile *dbfile, char *name);
```

*Arguments:*

dbfile	Database file pointer.
name	Object name.

*Returns:*

DBInqVarType returns the DBObjectType corresponding to the given object.

*Description:*

The DBInqVarType function returns the DBObjectType of the given object. The value returned is described in the following table:

Object Type	Returned Value
Invalid object or the object was not found in the file.	DB_INVALID_OBJECT
Quadmesh	DB_QUADMESH
Quadvar	DB_QUADVAR
UCD mesh	DB_UCDMESH
UCD variable	DB_UCDVAR
Multiblock mesh	DB_MULTIMESH
Multiblock variable	DB_MULTIVAR
Multiblock material	DB_MULTIMAT
Multiblock material species	DB_MULTIMATSPECIES
Material	DB_MATERIAL
Material species	DB_MATSPECIES
Facelist	DB_FACELIST
Zonelist	DB_ZONELIST
Edgelist	DB_EDGELIST
Curve	DB_CURVE
Pointmesh	DB_POINTMESH
Pointvar	DB_POINTVAR
Compound array	DB_ARRAY
Directory	DB_DIR

Object Type	Returned Value
Other variable (one written out using DBWrite.)	DB_VARIABLE
User-defined	DB_USERDEF

The function will signal an error if the given name does not exist in the file.

**DBMakeObject**—Allocate an object of the specified length and initialize it.

*Synopsis:*

```
DObject *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:*

```
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:*

```
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:*

```
DBfile *DBOpen (char *name, int type, int mode)
```

*Arguments:*

<code>name</code>	Name of the file to open. Can be either an absolute or relative path.
<code>type</code>	The type of file to open. One of the predefined types: <code>DB_SDX</code> , <code>DB_PDB</code> , <code>DB_TAURUS</code> , or <code>DB_UNKNOWN</code> .
<code>mode</code>	The mode of the file to open. One of the values <code>DB_READ</code> or <code>DB_APPEND</code> .

*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.

The mode parameter allows a user to append to an existing Silo file. If a file is DBOpen'ed with a mode of `DB_APPEND`, the file will support write operations as well as read operations.

**DBPause**—Pause a simulation*Synopsis:*

```
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:*

```
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 if 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.

**DBPutCurve**—Write a curve object into a Silo file*Synopsis:*

```
int DBPutCurve (DBfile *dbfile, char *curvename, void *xvals,
               void *yvals, int datatype, int npoints,
               DBoptlist *optlist)
```

*Arguments:*

<code>dbfile</code>	Database file pointer
<code>curvename</code>	Name of the curve object
<code>xvals</code>	Array of length <code>npoints</code> containing the x-axis data values
<code>yvals</code>	Array of length <code>npoints</code> containing the y-axis data values
<code>datatype</code>	Data type of the <code>xvals</code> and <code>yvals</code> arrays. One of the predefined Silo types.
<code>npoints</code>	The number of points in the curve
<code>optlist</code>	Pointer to an option list structure containing additional information to be included in the compound array object written into the Silo file. Use NULL if there are no options.

*Returns:*

DBPutCurve returns zero on success and -1 on failure.

*Description:*

The DBPutCurve function writes a curve object into a Silo file. A curve is a set of x/y points that describes a two-dimensional curve.

Both the `xvals` and `yvals` arrays must have the same datatype.

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_LABEL	int	Problem cycle value.	0
DBOPT_XLABEL	char *	Label for the x-axis	NULL
DBOPT_YLABEL	char *	Label for the y-axis	NULL
DBOPT_XUNITS	char *	Character string defining the units for the x-axis.	NULL
DBOPT_YUNITS	char *	Character string defining the units for the y-axis	NULL

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_XVARNAME	char *	Name of the domain (x) variable. This is the problem variable name, not the code variable name passed into the <code>xvals</code> argument.	NULL
DBOPT_YVARNAME	char *	Name of the domain (y) variable. This is problem variable name, not the code variable name passed into the <code>yvals</code> argument.	NULL

---

**DBPutFacelist**—Write a facelist object into a Silo file.

*Synopsis:*

```
int DBPutFacelist (DBfile *dbfile, char *name, int nfaces,
                  int ndims, int nodelist[], int lnodelist,
                  int origin, int zoneno[], int shapsize[],
                  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. ( <i>MeshTV requires a non-NULL zoneno for pseudocolor plots.</i> )
shapsize	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. Also note that MeshTV, in order to do pseudocolor plots correctly, requires a non-NULL zoneno.*

---

**DBPutMaterial**—Write a material data object into a Silo file.

*Synopsis:*

```
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:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the material data object.
<code>meshname</code>	Name of the mesh associated with this information.
<code>nmat</code>	Number of materials.
<code>matnos</code>	Array of length <code>nmat</code> containing material numbers.
<code>matlist</code>	Array whose dimensions are defined by <code>dims</code> and <code>ndims</code> . 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.
<code>dims</code>	Array of length <code>ndims</code> which defines the dimensionality of the <code>matlist</code> array.
<code>ndims</code>	Number of dimensions in <code>matlist</code> array.
<code>mix_next</code>	Array of length <code>mixlen</code> of indices into the mixed data arrays (one-origin).
<code>mix_mat</code>	Array of length <code>mixlen</code> of material numbers for the mixed zones.
<code>mix_zone</code>	Optional array of length <code>mixlen</code> of back pointers to originating zones. The origin is determined by <code>DBOPT_ORIGIN</code> . Even if <code>mixlen &gt; 0</code> , this argument is optional.
<code>mix_vf</code>	Array of length <code>mixlen</code> of volume fractions for the mixed zones.
<code>mixlen</code>	Length of mixed data arrays (or zero if no mixed data is present). If <code>mixlen &gt; 0</code> , then the “ <code>mix_</code> ” arguments describing the mixed data arrays must be non-NULL.
<code>datatype</code>	Volume fraction data type. One of the predefined Silo data types.
<code>optlist</code>	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_ORIGIN	int	Origin for mix_zone. Zero or one.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0
DBOPT_MATNAMES	char**	Array of strings defining the names of the individual materials.	NULL

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 index of the zone which contains this element. The origin is determined by DBOPT\_ORIGIN.

`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.

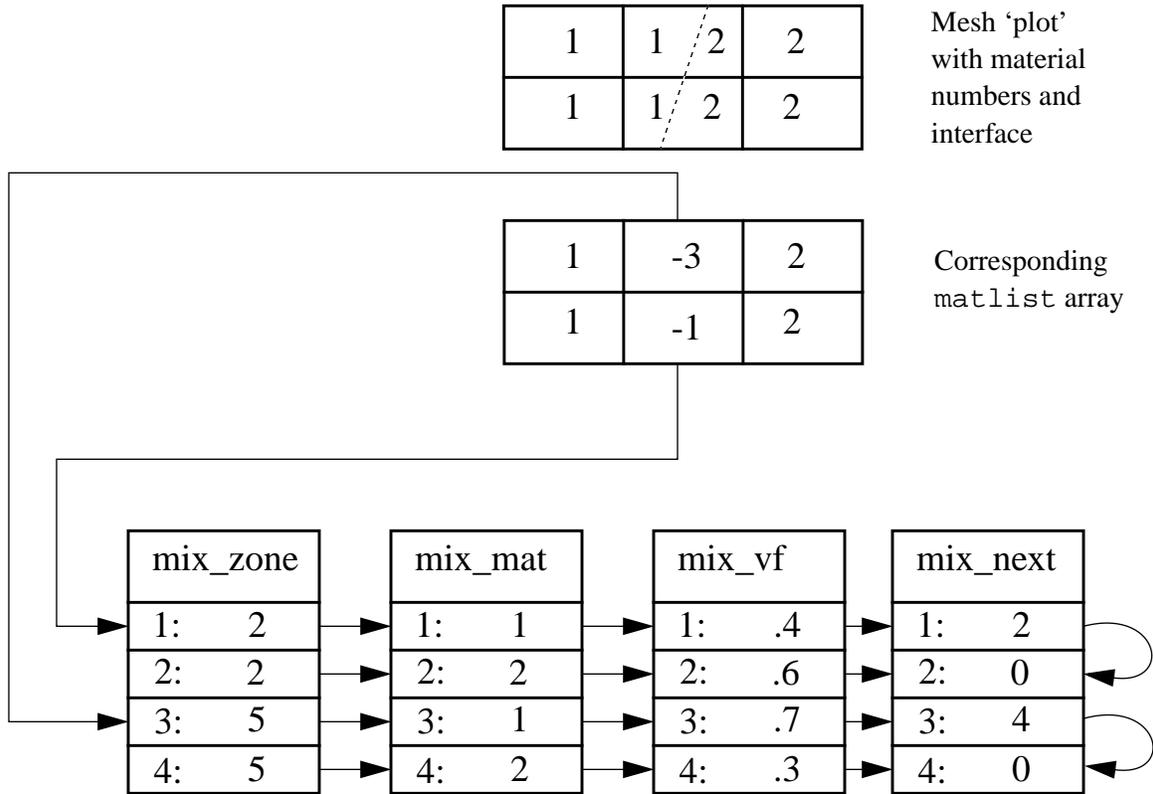


Figure 2-1: Example using mixed data arrays for representing material information

**DBPutMatspecies**—Write a material species data object into a Silo file.

*Synopsis:*

```
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:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the material species data object.
<code>matname</code>	Name of the material object with which the material species object is associated.
<code>nmat</code>	Number of materials.
<code>nmatspec</code>	Array of length <code>nmat</code> containing the number of material species associated with each material.
<code>speclist</code>	Array of dimension defined by <code>ndims</code> and <code>dims</code> of indices into the <code>species_mf</code> array. Each entry corresponds to one zone. If the zone is clean, the entry in this array must be positive or zero. A positive value is a 1-origin index to the mass fractions of the zone's material species. A zero can be used if the material in this zone contains only one species. If the zone is mixed, this value is ignored and the array <code>mix_list</code> is used instead.
<code>dims</code>	Array of length <code>ndims</code> that defines the length of the <code>speclist</code> array.
<code>ndims</code>	Number of dimensions in the <code>speclist</code> array.
<code>nspecies_mf</code>	Number of material species mass fractions.
<code>species_mf</code>	Array of length <code>nspecies_mf</code> containing mass fractions of the material species.
<code>mix_list</code>	Array of length <code>mixlen</code> containing indices into the <code>species_mf</code> array. These are used for mixed zones. For every index <code>j</code> in this array, <code>mix_list[j]</code> corresponds to the <code>DBmaterial</code> structure's material <code>mix_mat[j]</code> and zone <code>mix_zone[j]</code> .
<code>mixlen</code>	Length of the <code>mix_list</code> array.
<code>datatype</code>	The datatype of the mass fraction data in <code>species_mf</code> . One of the predefined Silo data types.
<code>optlist</code>	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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
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

**DBPutMultimat**—Write a multi-block material object into a Silo file.

*Synopsis:*

```
int DBPutMultimat (DBfile *dbfile, char *name, int nmat,
                  char *matnames[], DBoptlist *optlist)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the multi-material object.
<code>nmat</code>	Number of materials provided.
<code>matnames</code>	Array of length <code>nmat</code> containing pointers to the names of the materials to be associated with the multi-material object.
<code>optlist</code>	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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_BLOCKORIGIN	int	The origin of the block numbers.	1
DBOPT_GROUPORIGIN	int	The origin of the group numbers.	1
DBOPT_NGROUPS	int	The total number of groups in this multi-mat species object.	0
DBOPT_NMATNOS	int	Number of material numbers stored in the DBOPT_MATNOS option.	0
DBOPT_MATNOS	int *	Pointer to an array of length DBOPT_NMATNOS containing a complete list of the material numbers used in the Multimat object. DBOPT_NMATNOS must be set for this to work correctly.	NULL
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0

**DBPutMultimatpecies**—Write a multi-block species object into a Silo file.

*Synopsis:*

```
int DBPutMultimatpecies (DBfile *dbfile, char *name, int nspec,
                        char *specnames[], DBoptlist *optlist)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the multi-block species structure.
<code>nspec</code>	Number of species objects provided.
<code>specnames</code>	Array of length <code>nspec</code> containing pointers to the names of each of the species.
<code>optlist</code>	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:*

DBPutMultimatpecies returns zero on success and -1 on failure.

*Description:*

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

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_BLOCKORIGIN	int	The origin of the block numbers.	1
DBOPT_GROUPORIGIN	int	The origin of the group numbers.	1
DBOPT_NGROUPS	int	The total number of groups in this multi-mat species object.	0
DBOPT_MATNAME	char *	Character string defining the name of the multi-block material with which this object is associated.	NULL
DBOPT_NMAT	int	The number of materials in the associated material object.	0
DBOPT_NMATSPEC	int *	Array of length <code>DBOPT_NMAT</code> containing the number of material species associated with each material. <code>DBOPT_NMAT</code> must be set for this to work correctly.	NULL
DBOPT_CYCLE	int	Problem cycle value.	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

**DBPutMultimesh**—Write a multi-block mesh object into a Silo file.

*Synopsis:*

```
int DBPutMultimesh (DBfile *dbfile, char *name, int nmesh,
                   char *meshnames[], int meshtypes[],
                   DBoptlist *optlist)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the multi-block mesh structure.
<code>nmesh</code>	Number of meshes provided.
<code>meshnames</code>	Array of length <code>nmesh</code> containing pointers to the names of each of the meshes.
<code>meshtypes</code>	Array of length <code>nmesh</code> containing the type of each mesh. One of the predefined types: <code>DB_QUAD_RECT</code> , <code>DB_QUAD_CURV</code> , <code>DB_UCDMESH</code> , and <code>DB_POINTMESH</code> .
<code>optlist</code>	Pointer to an option list structure containing additional information to be included in the object written into the Silo file. Use a <code>NULL</code> 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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_BLOCKORIGIN	int	The origin of the block numbers.	1
DBOPT_GROUPORIGIN	int	The origin of the group numbers.	1
DBOPT_NGROUPS	int	The total number of groups in this multi-mesh object.	0
DBOPT_CYCLE	int	Problem cycle value.	0
DBOPT_TIME	float	Problem time value.	0.0
DBOPT_DTIME	double	Problem time value.	0.0

---

**DBPutMultivar**—Write a multi-block variable object into a Silo file.

*Synopsis:*

```
int DBPutMultivar (DBfile *dbfile, char *name, int nvar,
                  char *varnames[], int vartypes[],
                  DBoptlist *optlist);
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the multi-block variable.
<code>nvar</code>	Number of variables associated with the multi-block variable.
<code>varnames</code>	Array of length <code>nvar</code> containing pointers to the names of the variables. These are variables written with <code>DBPutPointvar</code> , <code>DBPutQuadvar</code> , and <code>DBPutUcdvar</code> .
<code>vartypes</code>	Array of length <code>nvar</code> containing the types of the variables. Each entry must be one of the following: <code>DB_POINTVAR</code> , <code>DB_QUADVAR</code> , or <code>DB_UCDVAR</code> .
<code>optlist</code>	Pointer to an option list structure containing additional information to be included in the object written into the Silo file. Use a <code>NULL</code> 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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
<code>DBOPT_BLOCKORIGIN</code>	int	The origin of the block numbers.	1
<code>DBOPT_GROUPORIGIN</code>	int	The origin of the group numbers.	1
<code>DBOPT_NGROUPS</code>	int	The total number of groups in this multivar object.	0
<code>DBOPT_CYCLE</code>	int	Problem cycle value.	0
<code>DBOPT_TIME</code>	float	Problem time value.	0.0
<code>DBOPT_DTIME</code>	double	Problem time value.	0.0

**DBPutPointmesh**—Write a point mesh object into a Silo file.

*Synopsis:*

```
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_GROUPNUM	int	The group number to which this point-mesh belongs.	-1 (not in a group)
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

---

Option Name	Value Data Type	Option Meaning	Default Value
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_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:*

```
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_NSSPACE	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:*

```
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_NSSPACE	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:*

```
int DBPutQuadmesh (DBfile *dbfile, char *name, char *coordnames[],
                  float *coords[], int dims[], int ndims,
                  int datatype, int coordtype,
                  DBoptlist *optlist)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the mesh.
<code>coordnames</code>	Array of length <code>ndims</code> containing pointers to the names to be provided when writing out the coordinate arrays. <i>This parameter is currently ignored and can be set as NULL.</i>
<code>coords</code>	Array of length <code>ndims</code> containing pointers to the coordinate arrays.
<code>dims</code>	Array of length <code>ndims</code> describing the dimensionality of the mesh. Each value in the <code>dims</code> array indicates the number of nodes contained in the mesh along that dimension.
<code>ndims</code>	Number of dimensions.
<code>datatype</code>	Datatype of the coordinate arrays. One of the predefined Silo data types.
<code>coordtype</code>	Coordinate array type. One of the predefined types: <code>DB_COLLINEAR</code> or <code>DB_NONCOLLINEAR</code> . Collinear coordinate arrays are always one-dimensional, regardless of the dimensionality of the mesh; non-collinear arrays have the same dimensionality as the mesh.
<code>optlist</code>	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_GROUPNUM	int	The group number to which this quad-mesh belongs.	-1 (not in a group)
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 <code>ndims</code> which defines zero-origin offsets from the last node for the ending index along each dimension.	{0,0,...}
DBOPT_LO_OFFSET	int *	Array of <code>ndims</code> 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_NSSPACE	int	Number of spatial dimensions used by this mesh.	<code>ndims</code>
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

---

Option Name	Value Data Type	Option Meaning	Default Value
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:*

```
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:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the variable.
<code>meshname</code>	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.
<code>nvars</code>	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.
<code>varnames</code>	Array of length <code>nvars</code> containing pointers to character strings defining the names associated with each sub-variable.
<code>vars</code>	Array of length <code>nvars</code> containing pointers to arrays defining the values associated with each subvariable
<code>dims</code>	Array of length <code>ndims</code> which describes the dimensionality of the variable. Each value in the <code>dims</code> array indicates the number of elements contained in the variable along that dimension.
<code>ndims</code>	Number of dimensions.
<code>mixvars</code>	Array of length <code>nvars</code> containing pointers to arrays defining the mixed-data values associated with each subvariable. If no mixed values are present, this should be NULL.
<code>mixlen</code>	Length of mixed data arrays, if provided.
<code>datatype</code>	Datatype of the variable. One of the predefined Silo data types.
<code>centering</code>	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
<code>optlist</code>	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 vari-

able 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
DBOPT_USESPECMF	int	Boolean (DB_OFF or DB_ON) value specifying whether or not to weight the variable by the species mass fraction when using material species data.	DB_OFF

**DBPutQuadvar1**— Write a scalar quad variable object into a Silo file.

*Synopsis:*

```
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
DBOPT_USESPECMF	int	Boolean (DB_OFF or DB_ON) value specifying whether or not to weight the variable by the species mass fraction when using material species data.	DB_OFF

**DBPutUcdmesh**—Write a UCD mesh object into a Silo file.

*Synopsis:*

```
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. <i>This parameter is currently ignored and can be set as NULL.</i>
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 DBPutFacelist]. 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.

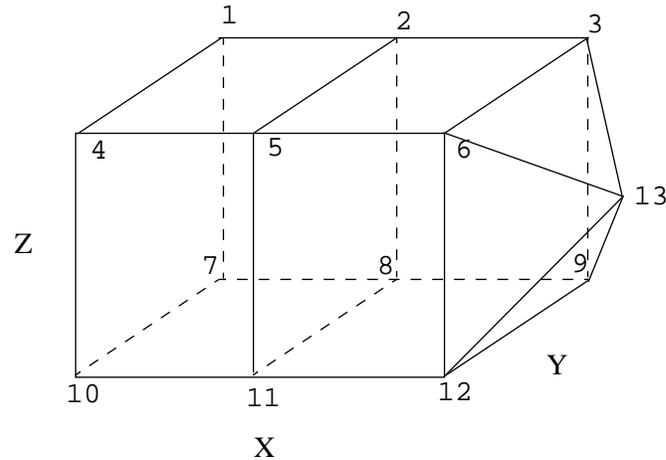
*Description:*

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:

See the description of “DBCalcExternalFacelist” on page 2-16 or “DBCalcExternalFacelist2” on page 2-17 for an automated way of computing the facelist needed for this call.



```

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,...}

fshapsize  = {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}

zshapsize  = {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. The node ordering for the predefined 3D cell types is illustrated below.

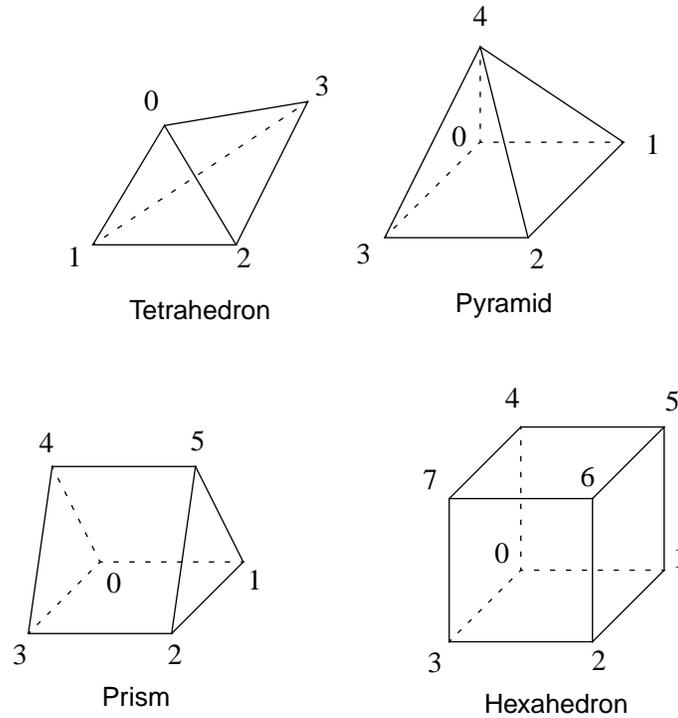
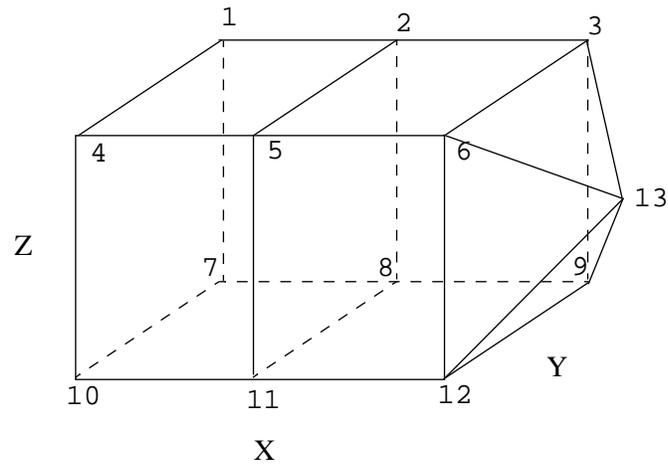


Figure 2-3: Node ordering for UCD zone shapes.

The nodes of a polyhedron are specified in the following fashion: First specify the number of faces in the polyhedron. Then, for each face, specify the number of nodes in the face followed by the nodes that make up the face. The nodes should be ordered such that they are numbered in a counter-clockwise fashion when viewed from the outside.



```

nzones      = 3
nzshapes     = 2
lznodelist  = 8 + 1 + 6 * 5 + 1 + 5 + 4 * 4 = 61
zodelist    = {1,4,5,2,7,10,11,8,
               6,
               4,11,12,9,8,
               4,12,6,3,9,
               4,6,5,2,3,
               4,5,11,8,2,
               4,5,6,12,11,
               4,3,2,8,9,
               5,
               4,3,6,12,9,
               3,6,13,12,
               3,12,13,9,
               3,9,13,3,
               3,3,13,6}
zshapetype  = {DB_ZONETYPE_HEX,
               DB_ZONETYPE_POLYHEDRON}
zshapecnt   = {1, 2}
zshapsize   = {8, 53}

```

Figure 2-4: Example usage of UCD zonelist combining a hex and 2 polyhedra.

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_GROUPNUM	int	The group number to which this UCD-mesh belongs.	-1 (not in a group)
DBOPT_COORDSYS	int	Coordinate system. One of: DB_CARTESIAN, DB_CYLINDRICAL, DB_SPHERICAL, DB_NUMERICAL, or DB_OTHER.	DB_OTHER
DBOPT_NODENUM	int*	An array of length <code>nnodes</code> giving a global node number for each node in the mesh.	NULL
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_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_NSSPACE	int	Number of spatial dimensions used by this mesh.	<code>ndims</code>
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:*

```
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:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the variable.
<code>meshname</code>	Name of the mesh associated with this variable (written with DBPutUcdmesh).
<code>nvars</code>	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.
<code>varnames</code>	Array of length <code>nvars</code> containing pointers to character strings defining the names associated with each subvariable.
<code>vars</code>	Array of length <code>nvars</code> containing pointers to arrays defining the values associated with each subvariable.
<code>nels</code>	Number of elements in this variable.
<code>mixvars</code>	Array of length <code>nvars</code> containing pointers to arrays defining the mixed-data values associated with each subvariable. If no mixed values are present, this should be NULL.
<code>mixlen</code>	Length of mixed data arrays (i.e., <code>mixvars</code> ).
<code>datatype</code>	Datatype of sub-variables. One of the predefined Silo data types.
<code>centering</code>	Centering of the sub-variables on the associated mesh. One of the predefined types: <code>DB_NODECENT</code> or <code>DB_ZONECENT</code> .
<code>optlist</code>	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
DBOPT_USESPECMF	int	Boolean (DB_OFF or DB_ON) value specifying whether or not to weight the variable by the species mass fraction when using material species data.	DB_OFF

**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:*

<code>dbfile</code>	Database file pointer.
<code>name</code>	Name of the variable.
<code>meshname</code>	Name of the mesh associated with this variable (written with either DBPutUcdmesh).
<code>var</code>	Array of length <code>nels</code> containing the values associated with this variable.
<code>nels</code>	Number of elements in this variable.
<code>mixvar</code>	Array of length <code>mixlen</code> containing the mixed-data values associated with this variable. If <code>mixlen</code> is zero, this value is ignored.
<code>mixlen</code>	Length of <code>mixvar</code> array. If zero, no mixed data is present.
<code>datatype</code>	Datatype of variable. One of the predefined Silo data types.
<code>centering</code>	Centering of the sub-variables on the associated mesh. One of the predefined types: DB_NODECENT or DB_ZONECENT.
<code>optlist</code>	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
DBOPT_USESPECMF	int	Boolean (DB_OFF or DB_ON) value specifying whether or not to weight the variable by the species mass fraction when using material species data.	DB_OFF

---

**DBPutZonelist**—Write a zonelist object into a Silo file.

*Synopsis:*

```
int DBPutZonelist (DBfile *dbfile, char *name, int nzones,
                  int ndims, int nodelist[], int lnodelist,
                  int origin, int shapsize[], 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.
shapsize	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 `zone1_name` parameter to the DBPutUcdmesh function.

*Notes:*

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

**DBPutZonelist2**—Write a zonelist object containing ghost zones into a Silo file.

*Synopsis:*

```
int DBPutZonelist2 (DBfile *dbfile, char *name, int nzones,
                   int ndims, int nodelist[], int lnodelist,
                   int origin, int lo_offset, int hi_offset,
                   int shapetype[], int shapysize[],
                   int shapecnt[], int nshapes,
                   DBoptlist *optlist)
```

*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.
lo_offset	The number of ghost zones at the beginning of the nodelist.
hi_offset	The number of ghost zones at the end of the nodelist.
shapetype	Array of length nshapes containing the type of each zone shape. See description below.
shapysize	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.
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:*

DBPutZonelist2 returns zero on success or -1 on failure.

*Description:*

The DBPutZonelist2 function writes a zonelist object into a Silo file. The name assigned to this object can in turn be used as the zone1\_name parameter to the DBPutUcdmesh function.

The allowed shape types are described in the following table:

Type	Description
DB_ZONETYPE_BEAM	A line segment
DB_ZONETYPE_POLYGON	A polygon where nodes are enumerated to form a polygon
DB_ZONETYPE_TRIANGLE	A triangle
DB_ZONETYPE_QUAD	A quadrilateral
DB_ZONETYPE_POLYHEDRON	A polyhedron with nodes enumerated to form faces and faces are enumerated to form a polyhedron
DB_ZONETYPE_TET	A tetrahedron
DB_ZONETYPE_PYRAMID	A pyramid
DB_ZONETYPE_PRISM	A prism
DB_ZONETYPE_HEX	A hexahedron

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_ZONENUM	int*	Array of global zone numbers, one per zone in this zonelist.	NULL

For a description of how the nodes for the allowed shapes are enumerated, see “DBPutUcdmesh” on page 2-98

**DBReadAtt**—Read an attribute value.

*Synopsis:*

```
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:*

```
int DBReadVar (DBfile *dbfile, char *varname, void *result)
```

*Arguments:*

dbfile	Database file pointer.
varname	Name of the simple variable.
result	Pointer to memory into which the 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 simple variable into the given space.

*Notes:*

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

**DBReadVar1**—Read one element from a simple variable.

*Synopsis:*

```
int DBReadVar1 (DBfile *dbfile, char *varname, int offset,  
               void *result)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>varname</code>	Name of the simple variable.
<code>offset</code>	Offset of one element to read.
<code>result</code>	Pointer to memory in which the 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 simple variable into the provided space.

**DBReadVarSlice**—Read a (hyper)slab of data from a simple variable.

*Synopsis:*

```
int DBReadVarSlice (DBfile *dbfile, char *varname, int *offset,
                  int *length, int *stride, int ndims,
                  void *result)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>varname</code>	Name of the simple variable.
<code>offset</code>	Array of length <code>ndims</code> of offsets in each dimension of the variable. This is the 0-origin position from which to begin reading the slice.
<code>length</code>	Array of length <code>ndims</code> of lengths of data in each dimension to read from the variable. All lengths must be positive.
<code>stride</code>	Array of length <code>ndims</code> of stride steps in each dimension. If no striding is desired, zeroes should be passed in this array.
<code>ndims</code>	Number of dimensions in the variable.
<code>result</code>	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 simple 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 one.

A one-dimensional array slice:

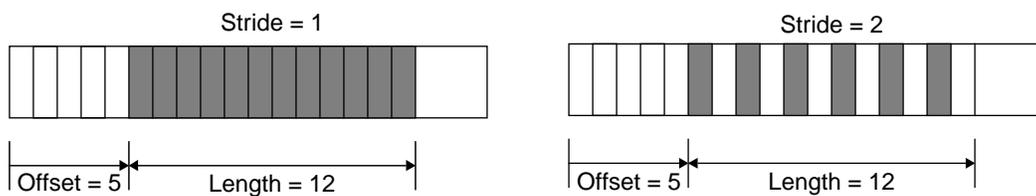


Figure 2-5: Array slice

**DBSetDataReadMask**—Set the data read mask*Synopsis:*

```
long DBSetDataReadMask (long mask)
```

*Arguments:*

**mask**                    The mask to use to read data. This is a bit vector of values that define whether each data portion of the various Silo objects should be read.

*Returns:*

DBSetDataReadMask returns the previous data read mask.

*Description:*

The DBSetDataReadMask allows the user to set the mask that's used to read the data within Silo objects.

Most Silo objects have a metadata portion and a data portion. The data portion is that part of the object that consists of pointers to long arrays of data. These arrays are "problem sized".

Setting the data read mask allows for a DBGet\* call to only return part of the data. With the data read mask set to DBAll, the DBGet\* functions return all of the information. With the data read mask set to DBNone, they return only the metadata. The mask is a bit vector specifying which part of the data model should be read.

A special case is found in the DBCalc flag. Sometimes data is not stored in the file, but is instead calculated from other information. The DBCalc flag controls this behavior. If it is turned off, the data is not calculated. If it is turned on, the data is calculated.

The values that DBSetDataReadMask takes as the mask parameter are binary-or'ed combinations of the values shown in the following table:

Mask bit	Meaning
DBAll	All data values are read. This value is identical to specifying all of the other mask bits or'ed together, setting all of the bit values to 1.
DBNone	No data values are read. This value sets all of the bit values to 0.
DBCalc	If data is calculable, calculate it. Otherwise, return NULL for that information.
DBMatMatnos	The lists of material numbers in material objects are read by the DBGetMaterial call.
DBMatMatnames	The arrays of material names in material objects are read by the DBGetMaterial call.
DBMatMatlist	The lists of the correspondence between zones and material numbers in material objects are read by the DBGetMaterial call.
DBMatMixList	The lists of mixed material information in material objects are read by the DBGetMaterial call.

Mask bit	Meaning
DBCurveArrays	The data values of curves are read by the DBGetCurve call.
DBPMCoords	The coordinate values of pointmeshes are read by the DBGetPointmesh call.
DBPVData	The data values of pointvars are read by the DBGetPointvar call.
DBQMCoords	The coordinate values of quadmeshes are read by the DBGetQuadmesh call.
DBQVData	The data values of quadvars are read by the DBGetQuadvar call.
DBUMCoords	The coordinate values of UCD meshes are read by the DBGetUcdmesh call.
DBUMFacelist	The facelists of UCD meshes are read by the DBGetUcdmesh call.
DBUMZonelist	The zonelists of UCD meshes are read by the DBGetUcdmesh call.
DBUVData	The data values of UCD variables are read by the DBGetUcdvar call.
DBFacelistInfo	The nodelists and shape information in facelists are read by the DBGetFacelist call.
DBZonelistInfo	The nodelist and shape information in zonelists are read by the DBGetZonelist call.

Use the DBGetDataReadMask call to retrieve the current data read mask without setting one.

Note: The data read mask is currently recognized only by the following drivers: PDB, Taurus.

**DBSetDir**—Set the current directory within the Silo database.

*Synopsis:*

```
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. Also, calls to DBSetDir will free the DBtoc structure, invalidating the pointer returned previously by DBGetToc. DBGetToc must be called again in order to obtain a pointer to the new directory's DBtoc structure.

---

**DBShowErrors**—Set the error reporting mode.

*Synopsis:*

```
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.

**DBVersion**—Get the version of the Silo library.

*Synopsis:*

```
char *DBVersion (void)
```

*Returns:*

DBVersion returns the version as a character string.

*Description:*

The DBVersion function determines what version of the Silo library is being used and returns that version in string form.

**DBWrite**—Write a simple variable.

*Synopsis:*

```
int DBWrite (DBfile *dbfile, char *varname, void *var, int *dims,
            int ndims, int datatype)
```

*Arguments:*

dbfile	Database file pointer.
varname	Name of the simple 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:*

```
int DBWriteComponent (DBfile *dbfile, DBOobject *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:*

```
int DBWriteObject (DBfile *dbfile, DBOBJECT *object, int freemem)
```

*Arguments:*

dbfile	Database file pointer.
object	Object created with DBMakeObject and populated with DBAddFltComponent, DBAddIntComponent, DBAddStrComponent, and DBAddVarComponent.
freemem	If non-zero, then the object will be freed after writing.

*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.

**DBWriteSlice**—Write a (hyper)slab of a simple variable*Synopsis:*

```
int DBWriteSlice (DBfile *dbfile, char *varname, void *var,
                 int datatype, int *offset, int *length,
                 int *stride, int *dims, int ndims)
```

*Arguments:*

<code>dbfile</code>	Database file pointer.
<code>varname</code>	Name of the simple variable.
<code>var</code>	Array defining the values associated with the slab.
<code>datatype</code>	Datatype of the variable. One of the predefined Silo data types.
<code>offset</code>	Array of length <code>ndims</code> of offsets in each dimension of the variable. This is the 0-origin position from which to begin writing the slice.
<code>length</code>	Array of length <code>ndims</code> of lengths of data in each dimension to write to the variable. All lengths must be positive.
<code>stride</code>	Array of length <code>ndims</code> of stride steps in each dimension. If no striding is desired, zeroes should be passed in this array.
<code>dims</code>	Array of length <code>ndims</code> which describes the dimensionality of the entire variable. Each value in the <code>dims</code> array indicates the number of elements contained in the entire variable along that dimension.
<code>ndims</code>	Number of dimensions.

*Returns:*

DBWriteSlice returns zero on success and -1 on failure.

*Description:*

The DBWriteSlice function writes a slab of data to a simple variable from the data provided in the `var` pointer. Any hyperslab of data may be written.

The size of the entire variable (after all slabs have been written) must be known when the DBWriteSlice function is called. The data in the `var` parameter is written into the entire variable using the location specified in the `offset`, `length`, and `stride` parameters. The data that makes up the entire variable may be written with one or more calls to DBWriteSlice.

The minimum `length` value is 1 and the minimum `stride` value is one.

A one-dimensional array slice:

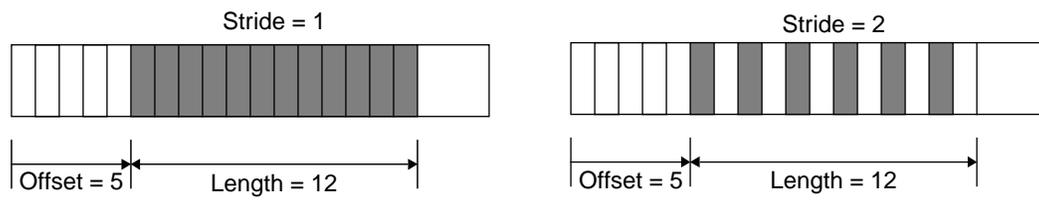


Figure 2-6: Array slice



**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 <code>abort()</code> 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 `siloinc.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
    optlist_id,                  ! to the list:
    option_id,                   ! the list
    int_value                     ! the option
)                                 ! 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
    dbputz1
    dbputfl
    dbputum
    dbputmat
```

---

dbputuv1  
.  
.  
.  
dbsetdir  
dbclose

**Table 3-1. Fortran Interface Functions and C Equivalents**

Fortran Functions	C Equivalent	Fortran Functions	C Equivalent
dbadd*opt	DBAddOption	dbputmmat	DBPutMultimat
dbcalfcl	DBCalfExternalFacelist	dbputmmesh	DBPutMultimesh
dbclose	DBCclose	dbputmsp	DBPutMatspecies
dbcreate	DBCreate	dbputmvar	DBPutMultivar
dbfgetca	—	dbputpm	DBPutPointmesh
dbfreeoptlist	DBFreeOptlist	dbputpv1	DBPutPointvar1
dbgetca	DBGetCompoundarray	dbputqm	DBPutQuadmesh
dbgetcurve	DBGetCurve	dbputqv1	DBPutQuadvar1
dbinqca	DBInqCompoundarray	dbputum	DBPutUcdmesh
dbinqfile	DBInqFile	dbputuv1	DBPutUcdvar1
dbinqlen	DBGetVarLength	dbputzl	DBPutZonelist
dbmkdir	DBMkDir	dbrdvar	DBReadVar
dbmkoptlist	DBMakeOptlist	dbrdvarslice	DBReadVarSlice
dbopen	DBOpen	dbsetdir	DBSetDir
dbputca	DBPutCompoundarray	dbshowerrors	DBShowErrors
dbputcurve	DBPutCurve	dbwrite	DBWrite
dbputfl	DBPutFacelist	dbwriteslice	DBWriteSlice
dbputmat	DBPutMaterial		

---

**dbadd\*opt**—Add an option to an option list.

*Synopsis:*

```
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 dbadd\*opt 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:*

```
integer dbcalcfl (zodelist, nnodes, origin, zshapsize,
                 zshapecnt, nzshapes, matlist, bnd_method, id)

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

*Arguments:*

<code>zodelist</code>	Array containing node indices describing mesh zones.
<code>nnodes</code>	Number of nodes in associated mesh.
<code>origin</code>	Origin for indices in the <code>zodelist</code> array. Should be zero or one.
<code>zshapsize</code>	Array of length <code>nzshapes</code> containing the number of nodes used by each zone shape.
<code>zshapecnt</code>	Array of length <code>nzshapes</code> containing the number of zones having each shape.
<code>nzshapes</code>	Number of zone shapes.
<code>matlist</code>	Array providing material numbers for each zone (else <code>DB_F77NULL</code> ).
<code>bnd_method</code>	Method to use for calculating external faces. See description below.
<code>id</code>	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:

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

**dbclose**—Close a Silo database.

*Synopsis:*

```
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:*

```
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_NO_CLOBBER.
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:*

```
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 <code>nvalues</code> containing the data values.
nvalues	Returned length of the <code>values</code> 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:*

```
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:*

```
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.

**dbgetcurve**—Read a curve from a Silo database.

*Synopsis:*

```
integer function dbgetcurve (dbid, curvename, lcurvename,  
                             maxpoints, xvals, yvals, datatype, npoints)  
  
integer dbid, lcurvename, maxpoints, datatype, npoints  
real(*) xvals, yvals  
character*(*) curvename
```

*Arguments:*

dbid	Database identifier.
curvename	Name of the curve to read.
lcurvename	Length of the curvename parameter in characters
maxpoints	The maximum number of points to read into the xvals and yvals arrays. Normally, this is the length of the arrays.
xvals	Array to read x-axis values into. No more than maxpoints values will be read into this array.
yvals	Array to read y-axis values into. No more than maxpoints values will be read into this array.
datatype	Returned Silo datatype of the data values
npoints	Returned number of points in the curve. Note that this number may be greater than maxpoints.

*Returns:*

dbcurve returns zero on success and -1 on failure..

*Description:*

The dbgetcurve function reads a curve from the Silo database and returns the data from that curve in the xvals, yvals, datatype, and npoints paramters..

**dbgetqv1**—Read a scalar quad variable object from a Silo database.

*Synopsis:*

```
integer function dbgetqv1 (dbid, varname, lvarname, var, dims,  
                          ndims, mixvar, mixlen, datatype, centering)  
  
integer dbid, lvarname, ndims, mixlen, datatype, centering  
integer dims(*)  
character*(*) varname  
real var(*), mixvar(*)
```

*Arguments:*

dbid	Database identifier.
varname	Name of the variable.
lvarname	Length of the name parameter in characters.
var	Returned array defining the values associated with this variable. The storage for the array must be supplied by the calling routine.
dims	Returned array of length <code>ndims</code> which describes the dimensionality of the variable. Each value in the <code>dims</code> array indicates the number of elements contained in the variable along that dimension. The storage for the array must be supplied by the calling routine.
ndims	Returned number of dimensions.
mixvar	Returned array defining the mixed-data values associated with this variable. The storage for the array must be supplied by the calling routine. If the mixed values are not desired, this should be set to <code>DB_F77NULL</code> .
mixlen	Returned length of the mixed data arrays.
datatype	Returned datatype of sub-variables. One of the predefined Silo data types.
centering	Returned centering of the sub-variables on the associated mesh. One of the predefined types: <code>DB_NODECENT</code> or <code>DB_ZONECENT</code> .

*Returns:*

`dbgetqv1` returns zero on success and -1 on failure.

*Description:*

The `dbgetqv1` function reads a scalar variable associated with a quad mesh from a Silo database.

**dbinqca**—Inquire Compound Array attributes.*Synopsis:*

```
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.

**dbinqfile**—Determine if filename is a Silo file.

*Synopsis:*

```
integer function dbinqfile (filename, lfilename, issilo)

integer lfilename, issilo
character*(*) filename
```

*Arguments:*

filename	Name of file.
lfilename	Length of the filename parameter in characters.
issilo	Returned value indicating if filename is a Silo file.

*Returns:*

dbinqfile returns zero on success and -1 on failure.

*Description:*

The dbinqfile function assigns zero to issilo if filename is not a Silo file and a non-zero number to issilo if filename is a Silo file.

**dbinqlen**—Return the number of elements in a simple variable.

*Synopsis:*

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

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

*Arguments:*

dbid	Database identifier.
varname	Simple 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 simple variable in number of elements. For example, a 16 byte array containing 4 floating point values has 4 elements.

**dbmkdir**—Create a new directory in the open Silo file.

*Synopsis:*

```
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:*

```
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 dbadd\*opt to populate the option list structure, and dbfreeoptlist to free it.

**dbopen**—Open an existing Silo file.

*Synopsis:*

```
integer function dbopen (name, lname, type, mode, dbid)

integer type, lname, mode, 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.
mode	The mode of the file to open. One of the values DB_READ or DB_APPEND.
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` is `DB_UNKNOWN`, Silo will guess at the file type, getting it right most of the time.

The mode parameter allows a user to append to an existing Silo file. If a file is dbopen'ed with a mode of `DB_APPEND`, the file will support write operations as well as read operations.

---

**dbputca**—Write a Compound Array object into a Silo file.

*Synopsis:*

```
integer function dbputca (dbid, name, lname, enames, lenames,
                        elengths, nelems, values, nvalues, datatype,
                        optlist_id, id)

integer dbid, lname, nelems, nvalues, datatype, optlist_id, 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 <code>nelems</code> containing the names of the elements. This is an array of 32-character strings.
lenames	Array of length <code>nelems</code> containing the lengths of the <code>enames</code> strings.
elengths	Array of length <code>nelems</code> containing the lengths of the elements.
nelems	Number of simple array elements.
values	Array whose length is determined by <code>nelems</code> and <code>elengths</code> containing the values of the simple array elements.
nvalues	Total length of the <code>values</code> array.
datatype	Data type of the <code>values</code> array. One of the predefined Silo types.
optlist_id	Option list identifier created with <code>dbmkoptlist</code> and populated with <code>dbadd*opt</code> . If no options are to be provided, use <code>DB_F77NULL</code> 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.

**dbputcurve**—Write a curve object into a Silo file*Synopsis:*

```
integer function dbputcurve (dbid, curvename, lcurvename, xvals,
                           yvals, datatype, npoints, optlist_id, id)

integer dbid, lcurvename, datatype, npoints, optlist_id, id
character*(*) curvename
real(*) xvals, yvals
```

*Arguments:*

dbid	Database identifier
curvename	Name of the curve
lcurvename	Length of the curvename parameter in characters
xvals	Array of length npoints containing the x-axis data values
yvals	Array of length npoints containing the y-axis data values
datatype	Data type of the xvals and yvals arrays. One of the predefined Silo types.
npoints	The number of points in the curve
optlist_id	Option list identifier created with dbmkoptlist and populated with dbadd*opt. 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.

*Returns:*

dbputcurve returns zero on success and -1 on failure.

*Description:*

The dbputcurve function writes a curve object into a Silo file. A curve is a set of x/y points that describe a two-dimensional curve.

Both the xvals and yvals arrays must have the same datatype.

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_LABEL	integer	Problem cycle value.	0
DBOPT_XLABEL	character*(*)	Label for the x-axis	NULL
DBOPT_YLABEL	character*(*)	Label for the y-axis	NULL
DBOPT_XUNITS	character*(*)	Character string defining the units for the x-axis.	NULL

---

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_YUNITS	character*(*)	Character string defining the units for the y-axis	
DBOPT_XVARNAME	character*(*)	Name of the domain (x) variable. This is the problem variable name, not the code variable name passed into the <code>xvals</code> argument.	NULL
DBOPT_YVARNAME	character*(*)	Name of the domain (y) variable. This is problem variable name, not the code variable name passed into the <code>yvals</code> argument.	NULL

**dbputfl**—Write a facelist object into a Silo file.

*Synopsis:*

```
integer function dbputfl (dbid, name, lname, nfaces, ndims,
                        nodelist, lnodelist, origin, zoneno,
                        shapsize, shapecnt, nshapes, types, typelist,
                        ntypes, id)

integer dbid, lname, nfaces, ndims, lnodelist, origin, nshapes
integer ntypes, id
character*(*) name
integer nodelist(*), shapsize(*), 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. ( <i>MeshTV requires a non-NULL zoneno for pseudocolor plots.</i> )
shapsize	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. Also note that MeshTV, in order to do pseudocolor plots correctly, requires a non-NULL zoneno.*

**dbputmat**—Write a material data object into a Silo file.

*Synopsis:*

```
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. The origin is determined by DBOPT_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 `dbadd*opt`. 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_ORIGIN	integer	Origin for <code>mix_zone</code> . Zero or one.	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 index of the zone which contains this element. The origin is determined by `DBOPT_ORIGIN`.

`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.

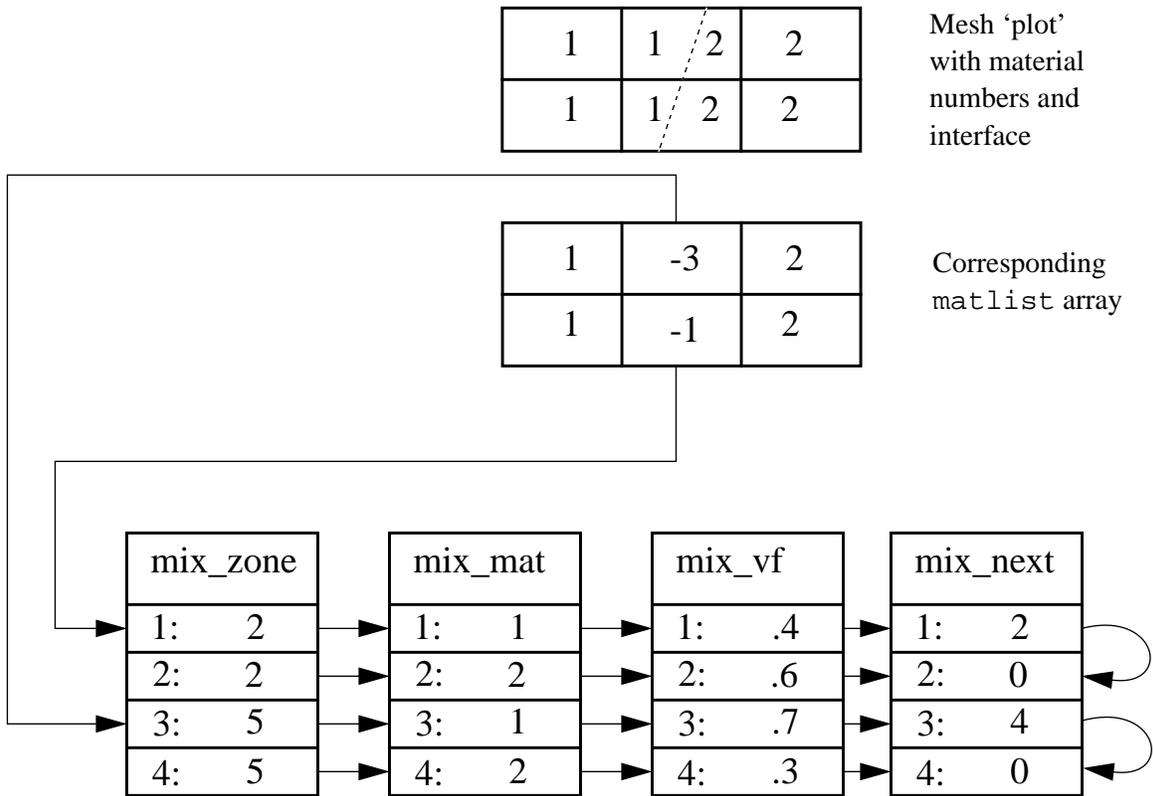


Figure 3-1: Example using mixed data arrays for representing material information.

**dbputmmat**—Write a multi-block material object into the open Silo file

*Synopsis:*

```
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 dbadd*opt. 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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_BLOCKORIGIN	integer	The origin of the block numbers.	1
DBOPT_GROUPORIGIN	integer	The origin of the group numbers.	1
DBOPT_NGROUPS	integer	The total number of groups in this multi-mat species object.	0
DBOPT_NMATNOS	integer	Number of material numbers stored in the DBOPT_MATNOS option.	0

---

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_MATNOS	integer*(*)	Array of length DBOPT_NMATNOS containing a complete list of the material numbers used in the Multimat object. DBOPT_NMATNOS must be set for this to work correctly.	NULL
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_TIME	real	Problem time value.	0.0

**dbputmmesh**—Write a multi-block mesh object into the open Silo file.

*Synopsis:*

```
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 dbadd*opt. 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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_BLOCKORIGIN	integer	The origin of the block numbers.	1
DBOPT_GROUPORIGIN	integer	The origin of the group numbers.	1

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_NGROUPS	integer	The total number of groups in this multi-mesh object.	0
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_TIME	real	Problem time value.	0.0

**dbputmsp**—Write a material species data object into a Silo file*Synopsis:*

```
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 dbadd*opt. 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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
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

**dbputmvar**—Write a multi-block variable object into the open Silo file.

*Synopsis:*

```
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 dbadd*opt. 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.

*Notes:*

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_BLOCKORIGIN	integer	The origin of the block numbers.	1
DBOPT_GROUPORIGIN	integer	The origin of the group numbers.	1
DBOPT_NGROUPS	integer	The total number of groups in this multivar object.	0

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_CYCLE	integer	Problem cycle value.	0
DBOPT_TIME	real	Problem time value.	0.0

**dbputpm**—Write a point mesh object into the open Silo file.

*Synopsis:*

```
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 dbadd*opt. 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_GROUPNUM	integer	The group number to which this pointmesh belongs.	-1 (not in a group)
DBOPT_CYCLE	integer	Problem cycle value.	0

---

Option Name	Value Data Type	Option Meaning	Default Value
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_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:*

```
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 dbadd*opt. 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_NSSPACE	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:*

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

```
integer dbid, lname, lxname, lname, 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 dbadd*opt. 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_GROUPNUM	integer	The group number to which this quadmesh belongs.	-1 (not in a group)
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 <i>ndims</i> 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 <i>ndims</i> 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_NSSPACE	integer	Number of spatial dimensions used by this mesh.	<i>ndims</i>
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:*

```
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 dbadd*opt. 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
DBOPT_USESPECMF	integer	Boolean (DB_OFF or DB_ON) value specifying whether or not to weight the variable by the species mass fraction when using material species data.	DB_OFF

**dbputum**—Write an UCD mesh object into the open Silo file.

*Synopsis:*

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

```
integer dbid, lname, lxname, lname, 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 dbadd*opt. 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.)

*Notes:*

See the description of “dbcalcf” on page 3-6 for an automated way of computing the facelist needed for this call.

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:

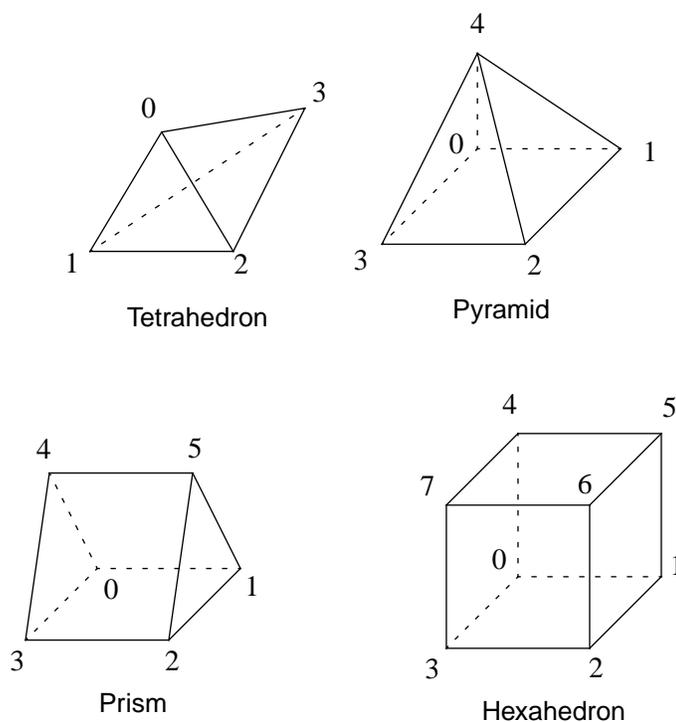
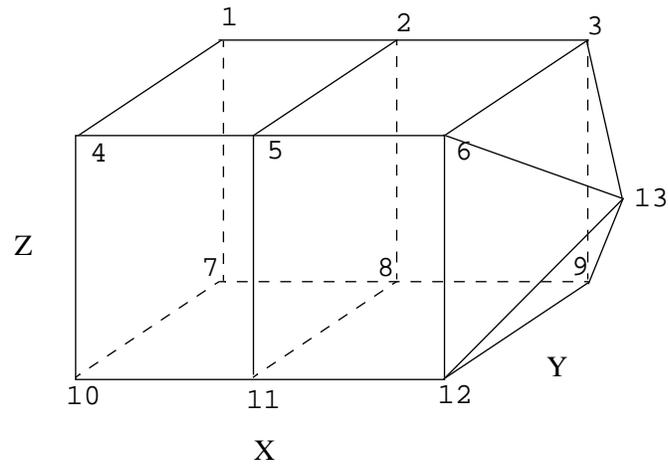


Figure 3-2: Node ordering for UCD zone shapes.



```

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,...}

fshapsize = {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}

zshapsize = {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 3-3: Example usage of UCD zonelist and external facelist variables.

The following table describes the options accepted by this function:

Option Name	Value Data Type	Option Meaning	Default Value
DBOPT_GROUPNUM	integer	The group number to which this UCDmesh belongs.	-1 (not in a group)
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
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_NSSPACE	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:*

```
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).
lmeshname	Length of the meshname parameter in characters.
var	Array of length <code>nels</code> containing the values associated with this variable.
nels	Number of elements in this variable.
mixvar	Array of length <code>mixlen</code> containing the mixed-data values associated with this variable. If <code>mixlen</code> is zero, this value is ignored.
mixlen	Length of the <code>mixvar</code> 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: <code>DB_NODECENT</code> or <code>DB_ZONECENT</code> .
optlist_id	Option list identifier created with <code>dbmkoptlist</code> and populated with <code>dbadd*opt</code> . If no options are to be provided, use <code>DB_F77NULL</code> 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
DBOPT_USESPECMF	integer	Boolean (DB_OFF or DB_ON) value specifying whether or not to weight the variable by the species mass fraction when using material species data.	DB_OFF

**dbputzl**—Write a zonelist object into a Silo file.

*Synopsis:*

```
integer function dbputzl (dbid, name, lname, nzones, ndims,  
                        nodelist, lnodelist, origin, shapsize,  
                        shapecnt, nshapes, idzl)  
  
integer dbid, lname, nzones, ndims, lnodelist, origin, nshapes,  
        idzl  
integer nodelist(*), shapsize(*), 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 <code>lnodelist</code> containing node indices describing mesh zones.
lnodelist	Length of the <code>nodelist</code> array.
origin	Origin for indices in the <code>nodelist</code> array. Should be zero or one.
shapsize	Array of length <code>nshapes</code> containing the number of nodes used by each zone shape.
shapecnt	Array of length <code>nshapes</code> 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 `zone1_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 variable.

*Synopsis:*

```
integer function dbrdvar (dbid, varname, lvarname, result)

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

*Arguments:*

dbid	Database identifier.
varname	Name of the simple variable
lvarname	Length of the varname parameter in characters.
result	Pointer to memory in which simple 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 variable into the given space.

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

*Synopsis:*

```
integer function dbrdvarslice (dbid, varname, lvarname, offset,
                             length, stride, ndims, dims, result)
```

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

*Arguments:*

dbid	Database identifier.
varname	Name of the simple variable.
lvarname	Length of the varname parameter in characters.
offset	Array of length ndims of offsets in each dimension of the entire 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 entire 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 entire variable.
result	Pointer to memory in which the slab of the 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:

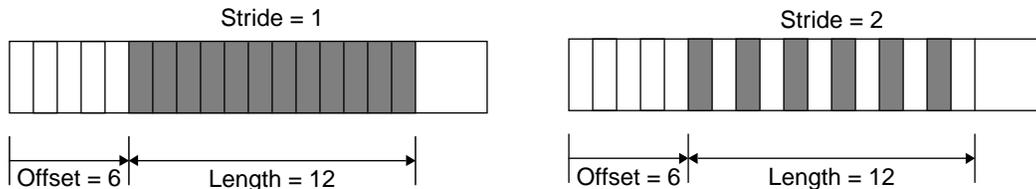


Figure 3-4: Array slice

**dbsetdir**—Set the current directory within the Silo database.

*Synopsis:*

```
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:*

```
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 variable.

*Synopsis:*

```
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 simple 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 simple 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.

**dbwriteslice**—Read a (hyper)slab of data from a simple variable.

*Synopsis:*

```
integer function dbwriteslice (dbid, varname, lvarname, var,
                             datatype, offset, length, stride, ndims)

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

*Arguments:*

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

*Returns:*

dbwriteslice returns zero on success and -1 on failure.

*Description:*

The dbwriteslice function writes a slab of data to a simple variable from the data provided in the var array. Any hyperslab of data may be read.

The size of the entire variable (after all slabs have been written) must be known when the dbwriteslice function is called. The data in the var parameter is written into the entire variable using the location specified in the offset, length, and stride parameters. The data that makes up the entire variable may be written with one or more calls to dbwriteslice.

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

A one-dimensional array slice:

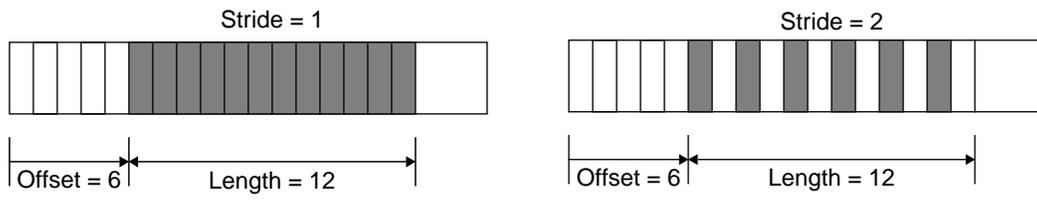


Figure 3-5: Array slice



## A.1 C Data Structures

### A.1.1 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.1.2 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 *yvarname;        /* Name of range (y) 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.1.3 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;        /* Use DB_ROWMAJOR to indicate row-major
                             for multi-d arrays. Use DB_COLMAJOR to
                             indicate column-major for multi-d array.s */

    int stride[3];          /* Offsets to adjacent elements in matlist*/
    int nmat;                /* Number of materials */
    int *matnos;             /* Array [nmat] of valid material numbers */
    char **matnames;        /* Array of material names */
    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.1.4 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;        /* Use DB_ROWMAJOR to indicate row-major
                             for multi-d arrays. Use DB_COLMAJOR to
                             indicate column-major for multi-d array.s */

    int stride[3];          /* Offsets to adjacent elements in 'speclist' */
    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. If the zone is clean, this array element will be a
                        positive index into the species_mf array. If the zone is
                        mixed, then this value is ignored. */

int mixlen;            /* Length of 'mix_speclist' array */
int *mix_speclist;    /* Array [mixlen] of 1-origin indices into the
                        'species_mf' array. mix_speclist[j] is the index in
                        array 'species_mf' of the first of the mass fractions
                        for material mix_mat[j] in zone mix_zone[j]*/

int datatype;         /* Datatype of mass fraction data */
} DBmatspecies;

```

### A.1.5 Point Variable Definition

```

typedef struct {
  /*----- Point 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;    /* Use DB_ROWMAJOR to indicate row-major
                        for multi-d arrays. Use DB_COLMAJOR to
                        indicate column-major for multi-d array.s */

  int stride[3];      /* Offsets to adjacent elements */
  int min_index[3];   /* Index in each dimension of 1st non-phony */
  int max_index[3];   /* Index in each dimension of last non-phony */
} DBmeshvar;

```

### A.1.6 Multi-Block Mesh Definition

```

typedef struct {
  /*----- Multi-Block Mesh -----*/
  int id;              /* Identifier for this object */
  int nblocks;         /* Number of blocks in mesh */
  int ngroups;         /* Number of groups 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 */
int blockorigin;         /* Origin (0 or 1) of block numbers */
int grouporigin;        /* Origin (0 or 1) of group numbers */
} DBmultimesh;

```

### A.1.7 Multi-Block Material Definition

```

typedef struct {
    /*----- Multi-Block Variable -----*/
    int id;                /* Identifier for this object */
    int nmats;             /* Number of materials */
    int ngroups;          /* Number of groups in mesh */
    char **matnames;      /* Material names */
    int blockorigin;      /* Origin (0 or 1) of block numbers */
    int grouporigin;      /* Origin (0 or 1) of group numbers */
} DBmultimat;

```

### A.1.8 Multi-Block Variable Definition

```

typedef struct {
    /*----- Multi-Block Variable -----*/
    int id;                /* Identifier for this object */
    int nvars;             /* Number of variables */
    int ngroups;          /* Number of groups in mesh */
    char **varnames;      /* Variable names */
    int *vartypes;        /* Variable types */
    int blockorigin;      /* Origin (0 or 1) of block numbers */
    int grouporigin;      /* Origin (0 or 1) of group numbers */
} DBmultivar;

```

### A.1.9 Multi-Block Species Definition

```

typedef struct {
    /*----- Multi-Block Variable -----*/
    int id;                /* Identifier for this object */
    int nspec;            /* Number of species objects */
    int ngroups;          /* Number of groups in mesh */
    char **specnames;     /* Names of species objects */
    int blockorigin;      /* Origin (0 or 1) of block numbers */
    int grouporigin;      /* Origin (0 or 1) of group numbers */
} DBmultimatspecies;

```

### A.1.10 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.1.11 Point Mesh Definition

```

typedef struct {
    /*----- Point Mesh-----*/
    int id;              /* Identifier for this object */
    int block_no;       /* Block number for this mesh */
    int group_no;       /* Group number for this mesh */
    char *name;         /* Name associated with mesh */
    int cycle;          /* Problem cycle number */
    float time;         /* Problem time */
    double dtime;       /* Problem time, double data type */
    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 */
    float min_extents[3]; /* Min mesh extents [ndims] */
    float max_extents[3]; /* Max mesh extents [ndims] */
    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.1.12 Quad Mesh Definition

```

typedef struct {
    /*----- Quad Mesh -----*/
    int id;              /* Identifier for this object */
    int block_no;       /* Block number for this mesh */
    int group_no;       /* Group number for this mesh */
    char *name;         /* Name associated with mesh */
    int cycle;          /* Problem cycle number */
    float time;         /* Problem time */
    double dtime;       /* Problem time, double date type */
    int coord_sys;      /* Cartesian, cylindrical, spherical */
    int major_order;    /* Use DB_ROWMAJOR to indicate row-major
                          for multi-d arrays. Use DB_COLMAJOR to
                          indicate column-major for multi-d array.s */

    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 */
int base_index[3];       /* Lowest real i,j,k value for this block */
int start_index[3];      /* i,j,k values corresponding to the original mesh */
int size_index[3];       /* # of nodes per dimension for original mesh */
} DBquadmesh;

```

### A.1.13 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 */
    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 ndims;               /* Rank of variable */
    int dims[3];             /* Number of elements in each dimension */
    int major_order;        /* Use DB_ROWMAJOR to indicate row-major
                               for multi-d arrays. Use DB_COLMAJOR to
                               indicate column-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. */

    int ascii_labels;       /* Treat variable values as ASCII values by rounding to
                               the nearest integer in the range [0, 255] */
} DBquadvar;

```

### A.1.14 Table of Contents Definiton

```

typedef struct {
    char    **curve_names      /* Array [ncurve] of pointers to curve names */
    int     ncurve            /* Number of curves */
    char    **multimesh_names /* Array [nmultimesh] of pointers to multimesh names
                               */
    int     nmultimesh       /* Number of multimeshes */
    char    **multivar_names  /* Array [nmultivar] of pointers to multi-variable
                               names */
    int     nmultivar        /* Number of multi-variables */
    char    **multimat_names  /* Array [nmultimat] of pointers to multi-material
                               names */
    int     nmultimat        /* Number of multi-variables */
    char    **multimatspecies_names /* Array [nmultimatspecies] of pointers to multi-
                               species names */
    int     nmultimatspecies  /* Number of multi-species */
    char    **qmesh_names    /* Array [nqmesh] of pointers to quadmesh names */
    int     nqmesh           /* Number of quadmeshes */
    char    **qvar_names     /* Array [nqvar] of pointers to quadmesh variable
                               names */
    int     nqvar            /* Number of quadmesh variables */
    char    **ucdmesh_names  /* Array [nucdmesh] of pointers to ucdmesh names */
    int     nucdmesh        /* Number of ucdmeshes */
    char    **ucdvar_names   /* Array [nucdvar] of pointers to ucdmesh variable
                               names */
    int     nucdvar         /* Number of ucdmesh variables */
    char    **ptmesh_names  /* Array [nptmesh] of pointers to pointmesh names */
    int     nptmesh         /* Number of pointmeshes */
    char    **ptvar_names   /* Array [nptvar] of pointers to pointmesh variable
                               names */
    int     nptvar          /* Number of pointmesh variables */
    char    **mat_names     /* Array [nmat] of pointers to materialnames */
    int     nmat            /* Number of materials */
    char    **matspecies_names /* Array [nmatspecies] of pointers to material species
                               names */
    int     nmatspecies     /* Number of material species */
    char    **var_names     /* Array [nvar] of pointers to variable names */
    int     nvar            /* Number of variables */
    char    **obj_names     /* Array [nobj] of pointers to object names */
    int     nobj            /* Number of objects */
    char    **dir_names     /* Array [ndir] of pointers to subdirectory names */
    int     ndir            /* Number of subdirectories */
    char    **array_names   /* Array [narrays] of pointers to array names */
    int     narrrays       /* Number of arrays */
} DBtoc;

```

### A.1.15 UCD Mesh Definition

```

typedef struct {

```

```

/*----- Unstructured Cell Data (UCD) Mesh -----*/
int id; /* Identifier for this object */
int block_no; /* Block number for this mesh */
int group_no; /* Group number for this mesh */
char *name; /* Name associated with mesh */
int cycle; /* Problem cycle number */
float time; /* Problem time */
double dtime; /* Problem time, double data type */
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) */
/* ----- Optional node attributes ----- */
int *nodeno /* [nnodes] Node number of each node */
int *gnodeno /* [nnodes] Global node number of each node */
} DBucdmesh;

```

#### A.1.16 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 */
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 '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) */
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. */
*/
}

```

```

    int ascii_labels;          /* Treat variable values as ASCII values by rounding to
                               the nearest integer in the range [0, 255] */
} DBucdvar;

```

#### A.1.17 UCD Edgelist Definition

```

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

```

#### A.1.18 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 occurrences of each shape */
    int *shapysize;        /* [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 *nodeno;           /* [lnodelist] node number of each node */
    int *zono;             /* [nfaces] Zone number for each face */
} DBfacelist;

```

#### A.1.19 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 occurrences of each shape */
    int *shapysize;        /* [nshapes] Number of nodes per shape */
    int *shapetype;         /* [nshapes] Type of shape */
    int *nodelist;          /* Sequential list of nodes which comprise zones */
    int lnodelist;          /* Number of nodes in nodelist */
    int origin;            /* '0' or '1' */
    int min_index;         /* Index of first real zone */
    int max_index;         /* Index of last real zone */
}

```

```
int *nodeno
int *gnodeno
} DBzonelist;

/* ----- Optional zone attributes ----- */
/* [nnodes] Node number of each node */
/* [nnodes] Global node number of each node */
```

---

# Glossary

---

<b>Block</b>	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.
<b>Curve</b>	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.
<b>Facelist</b>	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.
<b>Group</b>	A collection of blocks. Blocks within a group are usually structured meshes which can be logically indexed as though they comprised a single, larger block.
<b>Material</b>	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.
<b>Material species</b>	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.
<b>Mesh</b>	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.
<b>Multimat</b>	A set of materials. This object contains the names of the materials in the set.
<b>Multimesh</b>	A set of meshes. This object contains the names of and types of the meshes in the set.

---

<b>Multivar</b>	Mesh variable data associated with a multimesh.
<b>Node</b>	A mathematical point. The fundamental building-block of a mesh or zone.
<b>Pointmesh</b>	A point mesh. This includes dimension and coordinate data.
<b>Quadmesh</b>	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.
<b>Quadvar</b>	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.
<b>UCD</b>	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.
<b>Ucdmesh</b>	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.
<b>Ucdvar</b>	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.
<b>Variable</b>	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.
<b>Zone</b>	An area or volume from which meshes are comprised. Zones are polygons or polyhedra with nodes as vertices.
<b>Zonelist</b>	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.