

SAND92-2137  
Unlimited Release  
Printed November 1995

Distribution  
Category UC-705

## **EXODUS II: A Finite Element Data Model**

Larry A. Schoof, Victor R. Yarberr  
Computational Mechanics and Visualization Department  
Sandia National Laboratories  
Albuquerque, NM 87185

### **Abstract**

EXODUS II is a model developed to store and retrieve data for finite element analyses. It is used for preprocessing (problem definition), postprocessing (results visualization), as well as code to code data transfer. An EXODUS II data file is a random access, machine independent, binary file that is written and read via C, C++, or Fortran library routines which comprise the Application Programming Interface (API).

Intentionally Left Blank

# Table of Contents

	<u>Page</u>
1 Introduction .....	1
1.1 Availability .....	2
2 Development of EXODUS II .....	3
3 Description of Data Objects .....	4
3.1 Global Parameters .....	5
3.2 Quality Assurance Data .....	6
3.3 Information Data .....	6
3.4 Nodal Coordinates .....	6
3.4.1 Coordinate Names .....	6
3.5 Node Number Map .....	7
3.6 Element Number Map .....	7
3.7 Optimized Element Order Map .....	8
3.8 Element Blocks .....	8
3.8.1 Element Block Parameters .....	8
3.8.2 Element Connectivity .....	9
3.8.3 Element Attributes .....	9
3.9 Node Sets .....	11
3.9.1 Node Set Parameters .....	11
3.9.2 Node Set Node List .....	11
3.9.3 Node Set Distribution Factors .....	11
3.10 Concatenated Node Sets .....	11
3.11 Side Sets .....	12
3.11.1 Side Set Parameters .....	14
3.11.2 Side Set Element List .....	14
3.11.3 Side Set Side List .....	14
3.11.4 Side Set Node List .....	14
3.11.5 Side Set Node Count List .....	15
3.11.6 Side Set Distribution Factors .....	15
3.12 Concatenated Side Sets .....	15
3.13 Object Properties .....	17
3.13.1 Property Parameters .....	18
3.13.2 Property Values .....	18
3.14 Results Parameters .....	19
3.14.1 Results Names .....	19
3.15 Results Data .....	19
3.15.1 Time Values .....	19
3.15.2 Global Results .....	19
3.15.3 Nodal Results .....	19
3.15.4 Element Results .....	20
3.16 Element Variable Truth Table .....	20
4 Application Programming Interface (API) .....	21
4.1 Data File Utilities .....	22

4.1.1	Create EXODUS II File .....	23
4.1.2	Open EXODUS II File .....	25
4.1.3	Close EXODUS II File .....	27
4.1.4	Update EXODUS II File .....	28
4.1.5	Write Initialization Parameters .....	29
4.1.6	Read Initialization Parameters .....	31
4.1.7	Write QA Records .....	33
4.1.8	Read QA Records .....	35
4.1.9	Write Information Records .....	37
4.1.10	Read Information Records .....	39
4.1.11	Inquire EXODUS Parameters .....	41
4.1.12	Error Reporting .....	45
4.1.13	Set Error Reporting Level .....	47
4.2	Model Description .....	48
4.2.1	Write Nodal Coordinates .....	49
4.2.2	Read Nodal Coordinates .....	51
4.2.3	Write Coordinate Names .....	53
4.2.4	Read Coordinate Names .....	55
4.2.5	Write Node Number Map .....	56
4.2.6	Read Node Number Map .....	58
4.2.7	Write Element Number Map .....	59
4.2.8	Read Element Number Map .....	61
4.2.9	Write Element Order Map .....	62
4.2.10	Read Element Order Map .....	64
4.2.11	Write Element Block Parameters .....	65
4.2.12	Read Element Block Parameters .....	68
4.2.13	Read Element Blocks IDs .....	70
4.2.14	Write Element Block Connectivity .....	71
4.2.15	Read Element Block Connectivity .....	72
4.2.16	Write Element Block Attributes .....	73
4.2.17	Read Element Block Attributes .....	75
4.2.18	Write Node Set Parameters .....	77
4.2.19	Read Node Set Parameters .....	79
4.2.20	Write Node Set .....	81
4.2.21	Read Node Set .....	82
4.2.22	Write Node Set Distribution Factors .....	83
4.2.23	Read Node Set Distribution Factors .....	85
4.2.24	Read Node Sets IDs .....	86
4.2.25	Write Concatenated Node Sets .....	87
4.2.26	Read Concatenated Node Sets .....	91
4.2.27	Write Side Set Parameters .....	94
4.2.28	Read Side Set Parameters .....	96
4.2.29	Write Side Set .....	98
4.2.30	Read Side Set .....	100
4.2.31	Write Side Set Distribution Factors .....	102
4.2.32	Read Side Set Distribution Factors .....	104

4.2.33	Read Side Sets IDs .....	105
4.2.34	Read Side Set Node List .....	106
4.2.35	Write Concatenated Side Sets .....	108
4.2.36	Read Concatenated Side Sets .....	112
4.2.37	Convert Side Set Nodes to Sides .....	115
4.2.38	Write Property Arrays Names .....	118
4.2.39	Read Property Arrays Names .....	120
4.2.40	Write Object Property .....	122
4.2.41	Read Object Property .....	124
4.2.42	Write Object Property Array .....	126
4.2.43	Read Object Property Array .....	128
4.3	Results Data .....	130
4.3.1	Write Results Variables Parameters .....	131
4.3.2	Read Results Variables Parameters .....	133
4.3.3	Write Results Variables Names .....	135
4.3.4	Read Results Variables Names .....	137
4.3.5	Write Time Value for a Time Step .....	139
4.3.6	Read Time Value for a Time Step .....	141
4.3.7	Read All Time Values .....	143
4.3.8	Write Element Variable Truth Table .....	145
4.3.9	Read Element Variable Truth Table .....	147
4.3.10	Write Element Variable Values at a Time Step .....	149
4.3.11	Read Element Variable Values at a Time Step .....	152
4.3.12	Read Element Variable Values through Time .....	154
4.3.13	Write Global Variables Values at a Time Step .....	157
4.3.14	Read Global Variables Values at a Time Step .....	159
4.3.15	Read Global Variable Values through Time .....	161
4.3.16	Write Nodal Variable Values at a Time Step .....	163
4.3.17	Read Nodal Variable Values at a Time Step .....	165
4.3.18	Read Nodal Variable Values through Time .....	167
5	References .....	169
Appendix A	.....	A-1
Implementation of EXODUS II with netCDF	.....	A-1
Description	.....	A-1
Efficiency Issues	.....	A-1
netCDF Data Objects	.....	A-1
Attributes	.....	A-2
Dimensions	.....	A-2
Variables	.....	A-4
Appendix B	.....	B-1
Function Call Summary	.....	B-1
C binding routines	.....	B-1
FORTRAN binding routines	.....	B-9
Appendix C	.....	C-1
Error Messages	.....	C-1
Appendix D	.....	D-1

Sample Codes .....	D-1
C Write Example Code .....	D-1
C Read Example Code .....	D-10
FORTRAN Write Example Code .....	D-17
FORTRAN Read Example Code .....	D-24
Index .....	I-1

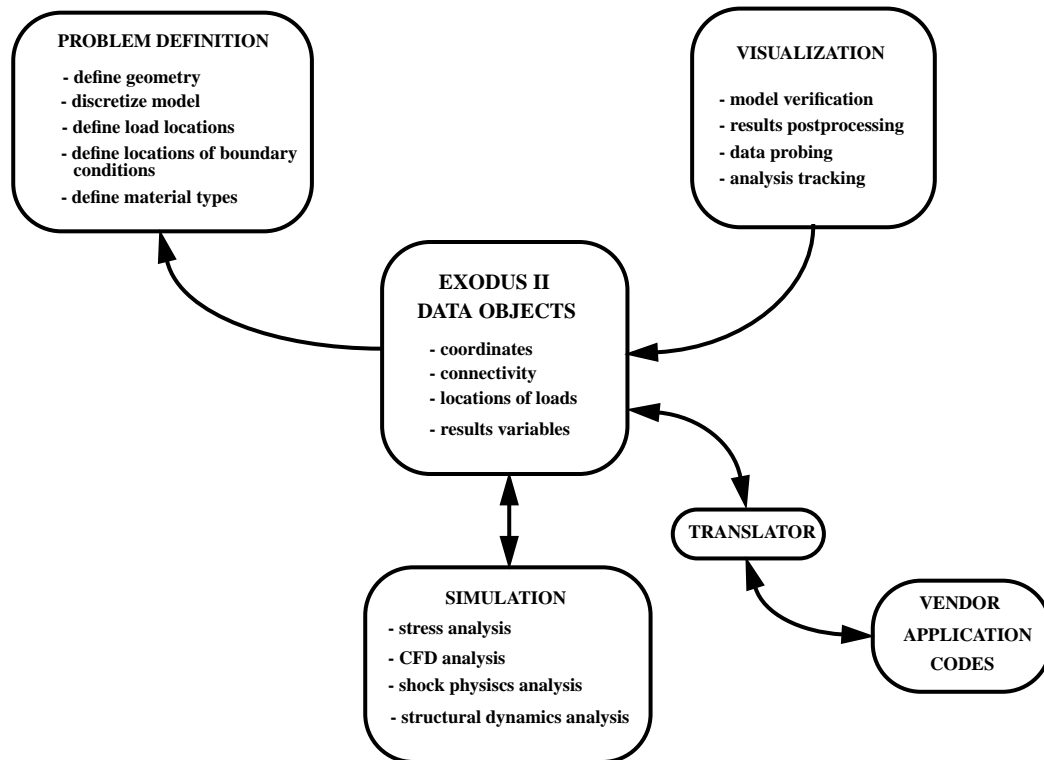
# 1 Introduction

EXODUS II is the successor of the widely used finite element (FE) data file format EXODUS [1] (henceforth referred to as EXODUS I) developed by Mills-Curran and Flanagan. It continues the concept of a common database for multiple application codes (mesh generators, analysis codes, visualization software, etc.) rather than code-specific utilities, affording flexibility and robustness for both the application code developer and application code user. By using the EXODUS II data model, a user inherits the flexibility of using a large array of application codes (including vendor-supplied codes) which access this common data file directly or via translators.

The uses of the EXODUS II data model include the following, as illustrated in Figure 1:

- problem definition -- mesh generation, specification of locations of boundary conditions and load application, specification of material types.
- simulation -- model input and results output.
- visualization -- model verification, results postprocessing, data interrogation, and analysis tracking.

**Figure 1** Uses of EXODUS II



## 1.1 Availability

The EXODUS II library is maintained in the Sandia National Laboratories Engineering Analysis Code Access System (SEACAS) [2] and is available on a licensed basis. For more information on obtaining the EXODUS II library, contact:

Marilyn K. Smith

Research Programs Department

Department 9103; MS 0833

Sandia National Laboratories

P.O. Box 5800

Albuquerque, NM 87185-0833

PHONE: (505) 844-3082

FAX: (505) 844-8251

EMAIL [mksmith@sandia.gov](mailto:mksmith@sandia.gov)

For bug reports, documentation errors, and enhancement suggestions, contact:

Larry A. Schoof

PHONE: (505) 844-5156

EMAIL: [laschoo@sandia.gov](mailto:laschoo@sandia.gov)



## 2 Development of EXODUS II

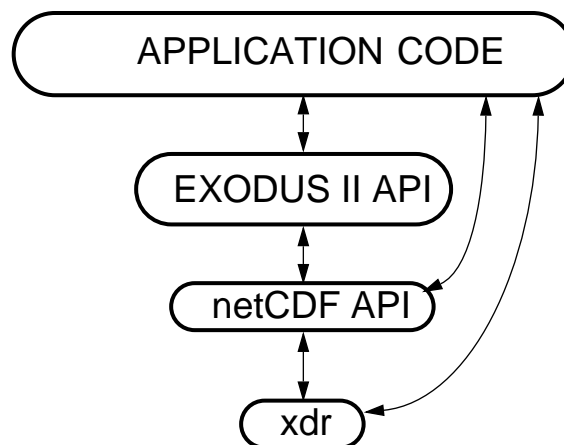
The evolution of the EXODUS data model has been steered by FE application code developers who desire the advantages of a common data format. The EXODUS II model has been designed to overcome deficiencies in the EXODUS I file format and meet the following functional requirements as specified by these developers:

- random read/write access.
- application programming interface (API) -- provide routines callable from FORTRAN, C, and C++ application codes.
- extensible -- allow new data objects to be added without modifying the application programs that use the file format.
- machine independent -- data should be independent of the machine which generated it.
- real time access during analysis -- allow access to the data in a file while the file is being created.

To address these requirements, the public domain database library netCDF [3] was selected to handle the low-level data storage. The EXODUS II library functions provide the mapping between FE data objects and netCDF dimensions, attributes, and variables. (These mappings are documented in Appendix A.) Thus, the code developer interacts with the data model using the vocabulary of an FE analyst (element connectivity, nodal coordinates, etc.) and is relieved of the details of the data access mechanism. To provide machine independency, the netCDF library stores data in eXternal Data Representation (XDR) [4] format.

Because an EXODUS II file is a netCDF file, an application program can access data via the EXODUS II API, the netCDF API, or XDR function calls directly. This functionality is illustrated in Figure 2. Although the latter two methods require more in-depth understanding of netCDF and/or XDR, this capability is a powerful feature that allows the development of auxiliary libraries of special purpose functions not offered in the standard EXODUS II library. For example, if an application required access to the coordinates of a single node (the standard library function returns the coordinates for all of the nodes in the model), a simple function could be written that calls netCDF routines directly to read the data of interest.

**Figure 2** EXODUS II Implementation



### 3 Description of Data Objects

The data in EXODUS II files can be divided into three primary categories: initialization data, model, and results.

Initialization data includes sizing parameters (number of nodes, number of elements, etc.), optional quality assurance information (names of codes that have operated on the data), and optional informational text.

The model is described by data which are static (do not change through time). This data includes nodal coordinates, element connectivity (node lists for each element), element attributes, and node sets and side sets (used to aid in applying loading conditions and boundary constraints).

The results are optional and include three types of variables -- nodal, element, and global - each of which is stored through time. Nodal results are output (at each time step) for all the nodes in the model. An example of a nodal variable is displacement in the X direction. Element results are output (at each time step) for all elements in one or more element blocks. For example, stress may be an element variable. Another use of element variables is to record element status (a binary flag indicating whether each element is “alive” or “dead”) through time. Global results are output (at each time step) for a single element or node, or for a single property. Linear momentum of a structure and the acceleration at a particular point are both examples of global variables. Although these examples correspond to typical FE applications, the data format is flexible enough to accommodate a spectrum of uses.

A few conventions and limitations must be cited:

- There are no restrictions on the frequency of results output except that the time value associated with each successive time step must increase monotonically.
- To output results at different frequencies (i.e., variable A at every simulation time step, variable B at every other time step) multiple EXODUS II files must be used.
- There are no limits to the number of each type of results, but once declared, the number cannot change.
- If the mesh geometry changes in time (i.e., number of nodes increases, connectivity changes), the new geometry must be output to a new EXODUS II file.

The following sections describe the data objects that can be stored in an EXODUS II file. API functions that read / write the particular objects are included for reference. API routines for the C binding are in lower case; functions for the Fortran binding are in upper case. Refer to Section 4 on page 21 for a detailed description of each API function.

## 3.1 Global Parameters

API functions: `ex_put_init`, `ex_get_init`; EXPINI, EXGINI

Every EXODUS II file is initialized with the following parameters:

- Title -- data file title of length `MAX_LINE_LENGTH` (`MXLNLN` in Fortran). Refer to discussion below for definition of `MAX_LINE_LENGTH`.
- Number of nodes -- the total number of nodes in the model.
- Problem dimension -- the number of spatial coordinates per node (1, 2, or 3).
- Number of elements -- the total number of elements of all types in the file.
- Number of element blocks -- within the EXODUS data model, elements are grouped together into blocks. Refer to Section 3.8 on page 8 for a description of element blocks.
- Number of node sets -- node sets are a convenient method for referring to groups of nodes. Refer to Section 3.9 on page 11 for a description of node sets.
- Number of side sets -- side sets are used to identify elements (and their sides) for specific purposes. Refer to Section 3.11 on page 12 for a description of side sets.
- Database version number -- the version of the data objects stored in the file. This document describes database version is 2.02.
- API version number -- the version of the EXODUS library functions which stored the data in the file. The API version can change without changing the database version and vice versa. This document describes API version 2.03.
- I/O word size -- indicates the precision of the floating point data stored in the file. Currently, four- or eight-byte floating point numbers are supported. It is not necessary that an application code be written to handle the same precision as the data stored in the file. If required, the routines in the EXODUS II library perform automatic conversion between four- and eight-byte numbers.
- Length of character strings -- all character data stored in an EXODUS II file is either of length `MAX_STR_LENGTH` (`MXSTLN` in Fortran) or `MAX_LINE_LENGTH` (`MXLNLN` in Fortran). This allows Fortran application codes to declare the lengths of character variables as predefined constants. These two constants are defined in the file `exodusII.h` (`exodusII.inc` for Fortran). Current values are 32 and 80, respectively.
- Length of character lines -- see description above for length of character strings.

## 3.2 Quality Assurance Data

API functions: `ex_put_qa`, `ex_get_qa`; `EXPQA`, `EXGQA`

Quality assurance (QA) data is optional information that can be included to indicate which application codes have operated on the data in the file. Any number of QA records can be included, with each record containing four character strings of length `MAX_STR_LENGTH` (`MXSTLN` in Fortran). The four character strings are the following (in order):

1. Code name -- indicates the application code that has operated on the EXODUS II file.
2. Code QA descriptor -- provides a location for a version identifier of the application code.
3. Date -- the date on which the application code was executed; should be in the format 01/25/93.
4. Time -- the 24-hour time at which the application code was executed; should be in the format hours:minutes:seconds, such as 16:30:15.

## 3.3 Information Data

API functions: `ex_put_info`, `ex_get_info`; `EXPINF`, `EXGINF`

This is for storage of optional supplementary text. Each text record is of length `MAX_LINE_LENGTH` (`MXLNLN` in Fortran); there is no limit to the number of text records.

## 3.4 Nodal Coordinates

API functions: `ex_put_coord`, `ex_get_coord`; `EXPCOR`, `EXGCON`

The nodal coordinates are the floating point spatial coordinates of all the nodes in the model. The number of nodes and the problem dimension define the length of this array. The node index cycles faster than the dimension index, thus the X coordinates for all the nodes is written before any Y coordinate data are written. Internal node numbers (beginning with 1) are implied from a nodes's place in the nodal coordinates record. See Section 3.5 on page 7 for a discussion of internal node numbers.

### 3.4.1 Coordinate Names

API functions: `ex_put_coord_names`, `ex_get_coord_names`; `EXPCON`,  
`EXGCON`

The coordinate names are character strings of length `MAX_STR_LENGTH` (`MXSTLN` in Fortran) which name the spatial coordinates. There is one string for each dimension in the model, thus there are one to three strings.

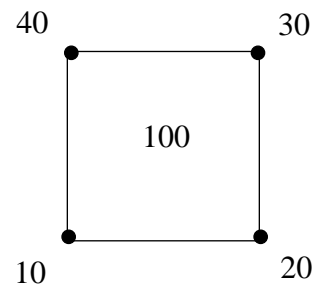
### 3.5 Node Number Map

API functions: `ex_put_node_num_map`, `ex_get_node_num_map`,  
`EXPNNM`, `EXGNNM`

Within the data model, internal node IDs are indices into the nodal coordinate array and internal element IDs are indices into the element connectivity array. Thus, internal node and element numbers (IDs) are contiguous (i.e., 1 . . . `number_of_nodes` and 1 . . . `number_of_elements`, respectively). Optional node and element number maps can be stored to relate user-defined node and element IDs to these internal node and element numbers. The length of these maps are `number_of_nodes` and `number_of_elements`, respectively. As an example, suppose a database contains exactly one QUAD element with four nodes. The user desires the element ID to be 100 and the node IDs to be 10, 20, 30, and 40 as shown in Figure 3.

**Figure 3** User-defined Node and Element IDs

Node IDs	Node coordinates	
10	0.0	0.0
20	1.0	0.0
30	1.0	1.0
40	0.0	1.0



The internal data structures representing the above model would be the following:

- nodal coordinate array: (0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0)
- connectivity array: (1, 2, 3, 4)
- node number map: (10, 20, 30, 40)
- element number map: (100)

Internal (contiguously numbered) node and element IDs must be used for all data structures that contain node or element numbers (IDs), including node set node lists, side set element lists, and element connectivity. Additionally, to inquire the value(s) of node or element results variables, an application code must pass the internal node or element number for the node or element of interest.

### 3.6 Element Number Map

API functions: `ex_put_elem_num_map`, `ex_get_elem_num_map`,  
`EXPENM`, `EXGENM`

Refer to Section 3.5 for a discussion of the optional element number map.

## 3.7 Optimized Element Order Map

API functions: `ex_put_map`, `ex_get_map`; `EXPMAP`, `EXGMAP`

The optional element order map defines the element order in which a solver (e.g., a wavefront solver) should process the elements. For example, the first entry is the number of the element which should be processed first by the solver. The length of this map is the total number of elements in the model.

## 3.8 Element Blocks

For efficient storage and to minimize I/O, elements are grouped into element blocks. Within an element block, all elements are of the same type (basic geometry and number of nodes). This definition does not preclude multiple element blocks containing the same element type (i.e., “QUAD” elements may be in more than one element block); only that each element block may contain only one element type.

The internal number of an element is defined implicitly by the order in which it appears in the file. Elements are numbered internally (beginning with 1) consecutively across all element blocks. See Section 3.5 on page 7 for a discussion of internal element numbering.

### 3.8.1 Element Block Parameters

API functions: `ex_put_elem_block`, `ex_get_elem_block`,  
`ex_get_elem_blk_ids`; `EXPELB`, `EXGELB`, `EXGEBI`

The following parameters are defined for each element block:

- Element block ID -- an arbitrary, unique, positive integer which identifies the particular element block. This ID is used as a “handle” into the database that allows users to specify a group of elements to the application code without having to know the order in which element blocks are stored in the file.
- Element type -- a character string of length `MAX_STR_LENGTH` (`MXSTLN` in Fortran) to distinguish element types. All elements within the element block are of this type. Refer to Table 1, “Element Types and Attributes,” on page 9 for a list of names that are currently accepted. For historical reasons, the names should be all upper case. It should be noted that the EXODUS II library routines do not verify element type names against a standard list; the interpretation of the element type is left to the application codes which read or write the data. In general, the first three characters uniquely identify the element type. Application codes can append characters to the element type string (up to the maximum length allowed) to further classify the element for specific purposes.
- Number of elements -- the number of elements in the element block.
- Nodes per element -- the number of nodes per element for the element block.

- Number of attributes -- the number of attributes per element in the element block. See below for a discussion of element attributes.

### 3.8.2 Element Connectivity

API functions: `ex_put_elem_conn`, `ex_get_elem_conn`; `EXPELC`, `EXGELC`

The element connectivity contains the list of nodes (internal node IDs; see Section 3.5 on page 7 for a discussion of node IDs) which define each element in the element block. The length of this list is the product of the number of elements and the number of nodes per element as specified in the element block parameters. The node index cycles faster than the element index. Node ordering follows the conventions illustrated in Figure 4, which includes ordering for higher order elements. For lower order elements, simply omit the unused nodes. These node ordering conventions follow the element topology used in PATRAN [5]. Thus, for higher order elements than those illustrated, use the ordering prescribed in the PATRAN User Manual. For elements of type `CIRCLE` or `SPHERE`, the topology is one node at the center of the circle or sphere element. .

### 3.8.3 Element Attributes

API functions: `ex_put_elem_attr`, `ex_get_elem_attr`; `EXPEAT`, `EXGEAT`

Element attributes are optional floating point numbers that can be assigned to each element. Every element in an element block must have the same number of attributes (as specified in the element block parameters) but the attributes may vary among elements within the block. The length of the attributes array is thus the product of the number of attributes per element and the number of elements in the element block. Table 1, “Element Types and Attributes,” lists the standard attributes for the given element types.

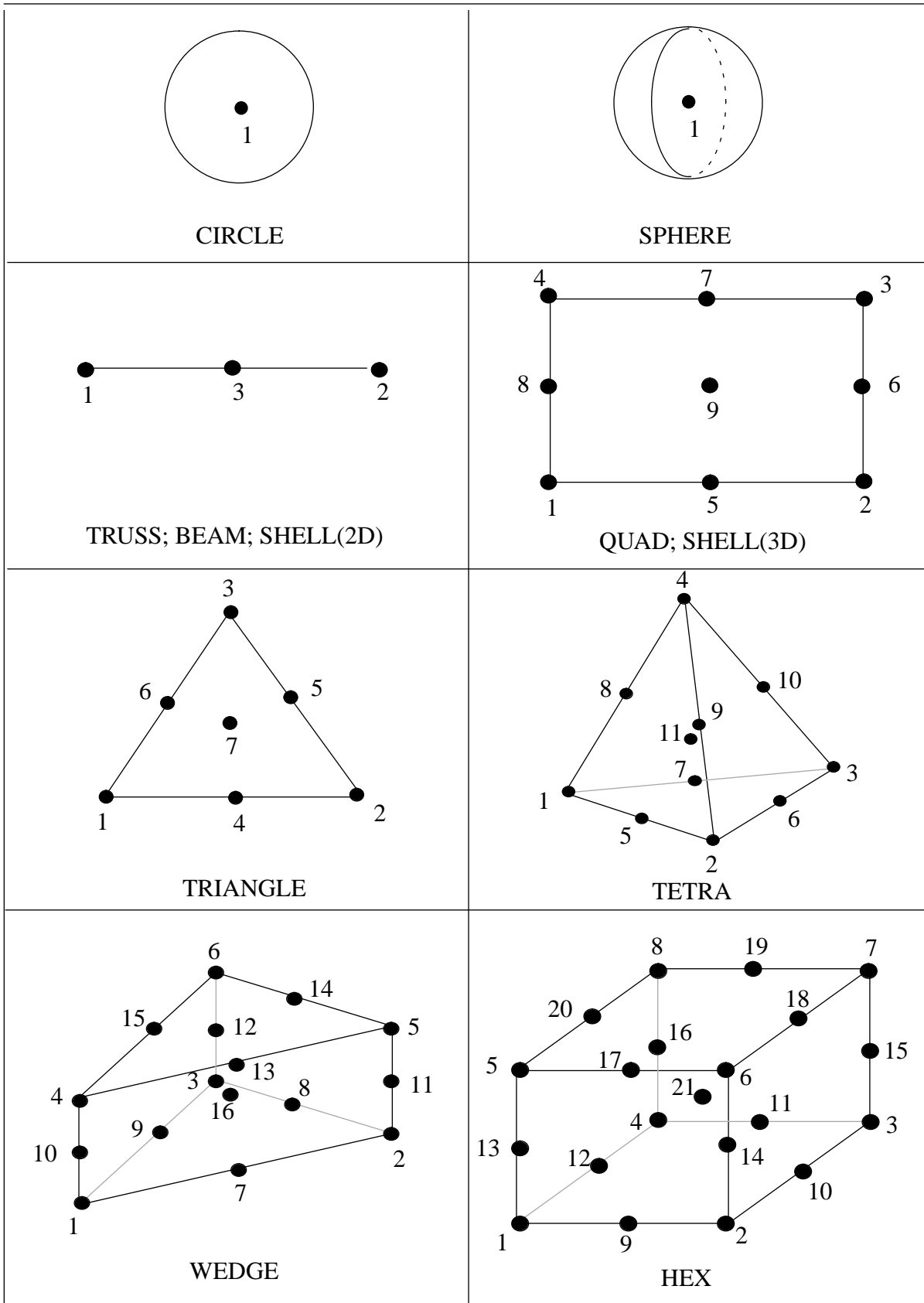
**Table 1** Element Types and Attributes

Element Type	Attributes
CIRCLE	R
SPHERE	R
TRUSS	A
BEAM	2D: A, I, J 3D: A, I <sub>1</sub> , I <sub>2</sub> , J, V <sub>1</sub> , V <sub>2</sub> , V <sub>3</sub>
TRIANGLE	
QUAD	
SHELL	T
TETRA	
WEDGE	
HEX	

#### Attribute Descriptions

- A -- cross-sectional area.
- V<sub>i</sub> -- a vector that, together with the axis of the element defines a plane for the beam element; I<sub>1</sub> bending moment of inertia affects displacements in this plane; I<sub>2</sub> bending moment of inertia affects bending out of this plane.
- J -- torsional (polar) moment of inertia.
- T -- thickness
- R -- radius

**Figure 4** Node Ordering for Standard Element Types





## 3.9 Node Sets

Node sets provide a means to reference a group of nodes with a single ID. Node sets may be used to specify load or boundary conditions, or to identify nodes for a special output request. A particular node may appear in any number of node sets, but may be in a single node set only once. (This restriction is not checked by EXODUS II routines.) Node sets may be accessed individually (using node set parameters, node set node list, and node set distribution factors) or in a concatenated format (described in Section 3.10 on page 11). The node sets data are stored identically in the data file regardless of which method (individual or concatenated) was used to output them.

### 3.9.1 Node Set Parameters

API functions: `ex_put_node_set_param`, `ex_get_node_set_param`,  
`ex_get_node_set_ids`; EXPNP, EXGNP, EXGNSI

The following parameters define each node set:

- Node set ID -- a unique integer that identifies the node set.
- Number of nodes -- the number of nodes in the node set.
- Number of distribution factors -- this should be zero if there are no distribution factors for the node set. If there are any distribution factors, this number must equal the number of nodes in the node set since the factors are assigned at each node. Refer to the discussion of distribution factors below.

### 3.9.2 Node Set Node List

API functions: `ex_put_node_set`, `ex_get_node_set`; EXPNS, EXGNS

This is an integer list of all the nodes in the node set. Internal node IDs (see Section 3.5 on page 7) must be used in this list.

### 3.9.3 Node Set Distribution Factors

API functions: `ex_put_node_set_dist_fact`,  
`ex_get_node_set_dist_fact`; EXPNSD, EXGNSD

This is an optional list of floating point factors associated with the nodes in a node set. These data may be used as multipliers on applied loads. If distribution factors are stored, each entry in this list is associated with the corresponding entry in the node set node list.

## 3.10 Concatenated Node Sets

API functions: `ex_put_concat_node_sets`,  
`ex_get_concat_node_sets`; EXPCNS, EXGCNS

Concatenated node sets provide a means of writing/reading all node sets with one function call. This is more efficient because it avoids some I/O overhead, particularly when considering the intricacies of the netCDF library. (Refer to Appendix A for a discussion of efficiency concerns.) This is accomplished with the following lists:

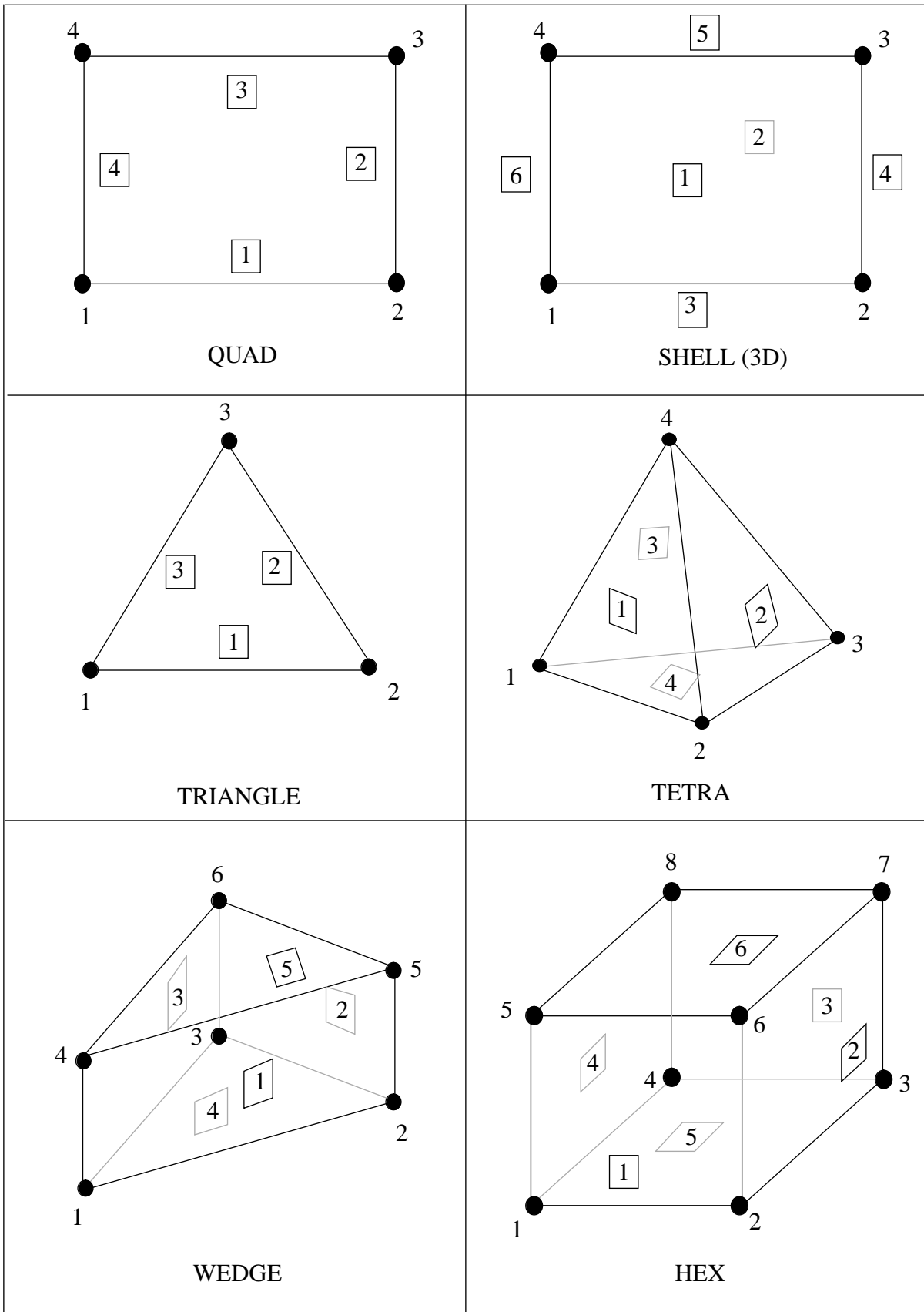
- Node sets IDs -- list (of length number of node sets) of unique integer node set ID's. The  $i$ th entry in this list specifies the ID of the  $i$ th node set.
- Node sets node counts -- list (of length number of node sets) of counts of nodes for each node set. Thus, the  $i$ th entry in this list specifies the number of nodes in the  $i$ th node set.
- Node sets distribution factors counts -- list (of length number of node sets) of counts of distribution factors for each node set. The  $i$ th entry in this list specifies the number of distribution factors in the  $i$ th node set.
- Node sets node pointers -- list (of length number of node sets) of indices which are pointers into the node sets node list locating the first node of each node set. The  $i$ th entry in this list is an index in the node sets node list where the first node of the  $i$ th node set can be located.
- Node sets distribution factors pointers -- list (of length number of node sets) of indices which are pointers into the node sets distribution factors list locating the first factor of each node set. The  $i$ th entry in this list is an index in the node sets distribution factors list where the first factor of the  $i$ th node set can be located.
- Node sets node list -- concatenated integer list of the nodes in all the node sets. Internal node IDs (see Section 3.5 on page 7) must be used in this list. The node sets node pointers and node sets node counts are used to find the first node and the number of nodes in a particular node set.
- Node sets distribution factors list -- concatenated list of the (floating point) distribution factors in all the node sets. The node sets distribution factors pointers and node sets distribution factors counts are used to find the first factor and the number of factors in a particular node set.

To clarify the use of these lists, refer to the coding examples in Section 4.2.25 and Section 4.2.26.

### 3.11 Side Sets

Side sets provide a second means of applying load and boundary conditions to a model. Unlike node sets, side sets are related to specified sides of elements rather than simply a list of nodes. For example, a pressure load must be associated with an element edge (in 2-d) or face (in 3-d) in order to apply it properly. Each side in a side set is defined by an element number and a local edge (for 2-d elements) or face (for 3-d elements) number. The local number of the edge or face of interest must conform to the conventions as illustrated in Figure 5. In this figure, side set side numbers are enclosed in boxes; only the essential node numbers to

**Figure 5** Side Set Side Numbering



describe the element topology are shown. A side set may contain sides of differing types of elements that are contained in different element blocks. For instance, a single side set may contain faces of WEDGE elements, HEX elements, and TETRA elements.

### 3.11.1 Side Set Parameters

API functions: `ex_put_side_set_param`, `ex_get_side_set_param`,  
`ex_get_side_set_ids`; EXPSP, EXGSP, EXGSSI

The following parameters define each side set:

- Side set ID -- a unique integer that identifies the side set.
- Number of sides -- the number of sides in the side set.
- Number of distribution factors -- this should be zero if there are no distribution factors for the side set. If there are any distribution factors, they are assigned at the nodes on the sides of the side set. Refer to the discussion of distribution factors below.

### 3.11.2 Side Set Element List

API functions: `ex_put_side_set`, `ex_get_side_set`; EXPSS, EXGSS

This is an integer list of all the elements in the side set. Internal element IDs (see Section 3.5 on page 7) must be used in this list.

### 3.11.3 Side Set Side List

API functions: `ex_put_side_set`, `ex_get_side_set`; EXPSS, EXGSS

This is an integer list of all the sides in the side set. This list contains the local edge (for 2-d elements) or face (for 3-d elements) numbers following the conventions specified in Figure 5.

### 3.11.4 Side Set Node List

API functions: `ex_get_side_set_node_list`; EXGSSN

It is important to note that the nodes on a side set are not explicitly stored in the data file, but can be extracted from the element numbers in the side set element list, local side numbers in the side set side list, and the element connectivity array. The node IDs that are output are internal node numbers (see Section 3.5 on page 7). They are extracted according to the following conventions:

1. All nodes for the first side (defined by the first element in the side set element list and the first side in the side set side list) are output before the nodes for the second side. There is no attempt to consolidate nodes; if a node is attached to four different faces, then the same node number will be output four times -- once each time the node is encountered when progressing along the side list.

2. The nodes for a single face (or edge) are ordered to assist an application code in determining an “outward” direction. Thus, the node list for a face of a 3-d element proceeds around the face so that the outward normal follows the right-hand rule. The node list for an edge of a 2-d element proceeds such that if the right hand is placed in the plane of the element palm down, thumb extended with the index (and other fingers) pointing from one node to the next in the list, the thumb points to the inside of the element. This node ordering is detailed in Table 2, “Side Set Node Ordering,” on page 16.
3. The nodes required for a first-order element are output first, followed by the nodes of a higher ordered element. Again, this is illustrated in Table 2, “Side Set Node Ordering,”

### 3.11.5 Side Set Node Count List

API functions: `ex_get_side_set_node_list`; EXGSSN

The length of the side set node count list is the length of the side set element list. For each entry in the side set element list, there is an entry in the side set side list, designating a local side number. The corresponding entry in the side set node count list is the number of nodes which define the particular side. In conjunction with the side set node list, this node count array provides an unambiguous nodal description of the side set.

### 3.11.6 Side Set Distribution Factors

API functions: `ex_put_side_set_dist_fact`,  
`ex_get_side_set_dist_fact`; EXPSSD, EXGSSD

This is an optional list of floating point factors associated with the nodes on a side set. These data may be used for uneven application of load or boundary conditions. Because distribution factors are assigned at the nodes, application codes that utilize these factors must read the side set node list. The distribution factors must be stored/accessed in the same order as the nodes in the side set node list; thus, the ordering conventions described above apply.

## 3.12 Concatenated Side Sets

API functions: `ex_put_concat_side_sets`,  
`ex_get_concat_side_sets`; EXPCSS, EXGCSS

Concatenated side sets provide a means of writing / reading all side sets with one function call. This is more efficient because it avoids some I/O overhead, particularly when considering the intricacies of the netCDF library. This is accomplished with the following lists:

- Side sets IDs -- list (of length number of side sets) of unique integer side set ID's. The *i*th entry in this list specifies the ID of the *i*th side set.

**Table 2** Side Set Node Ordering

Element Type	Side #	Node Order
QUAD	1	1, 2, 5
	2	2, 3, 6
	3	3, 4, 7
	4	4, 1, 8
SHELL	1	1, 2, 3, 4, 5, 6, 7, 8, 9
	2	4, 3, 2, 1, 7, 6, 5, 8, 9
	3	1, 2, 5
	4	2, 3, 6
	5	3, 4, 7
	6	4, 1, 8
TRIANGLE	1	1, 2, 4
	2	2, 3, 5
	3	3, 1, 6
TETRA	1	1, 2, 4, 5, 9, 8
	2	2, 3, 4, 6, 10, 9
	3	1, 4, 3, 8, 10, 7
	4	1, 3, 2, 7, 6, 5
WEDGE	1	1, 2, 5, 4, 7, 11, 13, 10
	2	2, 3, 6, 5, 8, 12, 14, 11
	3	1, 4, 6, 3, 10, 15, 12, 9
	4	1, 3, 2, 9, 8, 7
	5	4, 5, 6, 13, 14, 15
HEX	1	1, 2, 6, 5, 9, 14, 17, 13
	2	2, 3, 7, 6, 10, 15, 18, 14
	3	3, 4, 8, 7, 11, 16, 19, 15
	4	1, 5, 8, 4, 13, 20, 16, 12
	5	1, 4, 3, 2, 12, 11, 10, 9
	6	5, 6, 7, 8, 17, 18, 19, 20

- Side sets side counts -- list (of length number of side sets) of counts of sides for each side set. Thus, the  $i$ th entry in this list specifies the number of sides in the  $i$ th node set. This also defines the number of elements in each side set.
- Side sets distribution factors counts -- list (of length number of side sets) of counts of distribution factors for each side set. The  $i$ th entry in this list specifies the number of distribution factors in the  $i$ th side set.
- Side sets side pointers -- list (of length number of side sets) of indices which are pointers into the side sets element list (and side list) locating the first element (or side) of each side set. The  $i$ th entry in this list is an index in the side sets element list (and side list) where the first element (or side) of the  $i$ th side set can be located.
- Side sets distribution factors pointers -- list (of length number of side sets) of indices which are pointers into the side sets distribution factors list locating the first factor of each side set. The  $i$ th entry in this list is an index in the side sets distribution factors list where the first factor of the  $i$ th side set can be located.
- Side sets element list -- concatenated integer list of the elements in all the side sets. Internal element IDs (see Section 3.5 on page 7) must be used in this list. The side sets side pointers and side sets side counts are used to find the first element and the number of elements in a particular side set.
- Side sets side list -- concatenated integer list of the sides in all the side sets. The side sets side pointers and side sets side counts are used to find the first side and the number of sides in a particular side set.
- Side sets distribution factors list -- concatenated list of the (floating point) distribution factors in all the side sets. The side sets distribution factors pointers and side sets distribution factors counts are used to find the first factor and the number of factors in a particular side set.

### 3.13 Object Properties

Certain EXODUS II objects (currently element blocks, node sets, and side sets) can be given integer properties, providing the following capabilities:

1. assign a specific integer value to a named property of an object.
2. tag objects as members of a group. For example element blocks 1 and 3 and side sets 1 and 2 could be put in a group named "TOP."

This functionality is illustrated in Table 3, "Sample Property Table," which contains the property values of a sample EXODUS II file with three element blocks, one node set, and two side sets. Note that an application code can define properties to be valid for only specified object types. In this example, "STEEL" and "COPPER" are valid for all element blocks but are not defined for node sets and side sets.

**Table 3** Sample Property Table

NAME	EB 1	EB 2	EB 3	NS 1	SS 1	SS 2
ID	10	20	30	100	200	201
TOP	1	0	1	0	1	1
LEFT	1	1	0	1	1	0
STEEL	0	0	1	NULL	NULL	NULL
COPPER	1	1	0	NULL	NULL	NULL

Interpretation of the integer values of the properties is left to the application codes, but in general, a nonzero positive value means the object has the named property (or is in the named group); a zero means the object does not have the named property (or is not in the named group). Thus, element block 1 has an ID of 10 (1 is a counter internal to the data base; an application code accesses the element block using the ID), node set 1 has an ID of 100, etc. The group “TOP” includes element block 1, element block 3, and side sets 1 and 2.

### 3.13.1 Property Parameters

API functions: `ex_put_prop_names`, `ex_get_prop_names`; `EXPPN`, `EXGPN`

The parameters include the number of properties and the names of length `MAX_STR_LENGTH` (`MXSTLN` in Fortran) for each property for each object type (i.e., element blocks, node sets, or side sets). In the preceding example, there are five properties for element blocks (i.e., “ID”, “TOP”, “LEFT”, “STEEL”, and “COPPER”), three properties for node sets (i.e., “ID”, “TOP”, and “LEFT”), and three properties for side sets (i.e., “ID”, “TOP”, and “LEFT”).

### 3.13.2 Property Values

API functions: `ex_put_prop`, `ex_get_prop`, `ex_put_prop_array`,  
`ex_get_prop_array`; `EXPP`, `EXGP`, `EXPPA`, `EXGPA`

Valid values for the properties are positive integers and zero. Property values are stored in arrays in the data file but can be written / read individually given an object type (i.e., element block, node set, or side set), object ID, and property name or as an array given an object type and property name. If accessed as an array, the order of the values in the array must correspond to the order in which the element blocks, node sets, or side sets were introduced into the file. For instance, if the parameters for element block with ID 20 were written to a file, and then parameters for element block with ID 10, followed by the parameters for element block with ID 30, the first, second, and third elements in the property array would correspond to element block 20, element block 10, and element block 30, respectively. This order can be determined with a call to `ex_get_elem_blk_ids` (`EXGEBI` for Fortran) which returns an



array of element block IDs in the order that the corresponding element blocks were introduced to the data file.

### 3.14 Results Parameters

API functions: `ex_put_var_param`, `ex_get_var_param`; EXPVP, EXGVP

The number of each type of results variables (element, nodal, and global) is specified only once, and cannot change through time.

#### 3.14.1 Results Names

API functions: `ex_put_var_names`, `ex_get_var_names`; EXPVAN, EXGVAN

Associated with each results variable is a unique name of length MAX\_STR\_LENGTH (MXSTLN in Fortran).

### 3.15 Results Data

An integer output time step number (beginning with 1) is used as an index into the results variables written to or read from an EXODUS II file. It is a counter of the number of “data planes” that have been written to the file. The maximum time step number (i.e., the number of time steps that have been written) is available via a call to the database inquire function (See “Inquire EXODUS Parameters” on page 41). For each output time step, the following information is stored.

#### 3.15.1 Time Values

API functions: `ex_put_time`, `ex_get_time`, `ex_get_all_times`; EXPTIM, EXGTIM, EXGATM

A floating point value must be stored for each time step to identify the “data plane.” Typically, this is the analysis time but can be any floating point variable that distinguishes the time steps. For instance, for a modal analysis, the natural frequency for each mode may be stored as a “time value” to discriminate the different sets of eigen vectors. The only restriction on the time values is that they must monotonically increase.

#### 3.15.2 Global Results

API functions: `ex_put_glob_vars`, `ex_get_glob_vars`,  
`ex_get_glob_var_time`; EXPGV, EXGGV, EXGGVT

This object contains the floating point global data for the time step. The length of the array is the number of global variables, as specified in the results parameters.

#### 3.15.3 Nodal Results

API functions: `ex_put_nodal_var`, `ex_get_nodal_var`,  
`ex_get_nodal_var_time`; EXPNV, EXGNV, EXGNVT

This object contains the floating point nodal data for the time step. The size of the array is the number of nodes, as specified in the global parameters, times the number of nodal variables.

### 3.15.4 Element Results

API functions: `ex_put_elem_var`, `ex_get_elem_var`,  
`ex_get_elem_var_time`; `EXPEV`, `EXGEV`, `EXGEVT`

Element variables are output for a given element block and a given element variable. Thus, at each time step, up to  $m$  element variable objects (where  $m$  is the product of the number of element blocks and the number of element variables) may be stored. However, since not all element variables must be output for all element blocks (see Element Variable Truth Table below),  $m$  is the *maximum* number of element variable objects. The actual number of objects stored is the number of unique combinations of element variable index and element block ID passed to `ex_put_elem_var` (`EXPEV` for Fortran) or the number of non-zero entries in the element variable truth table (if it is used). The length of each object is the number of elements in the given element block.

### 3.16 Element Variable Truth Table

API functions: `ex_put_elem_var_tab`, `ex_get_elem_var_tab`; `EXPVTT`,  
`EXGVTT`

Because some element variables are not applicable (and thus not computed by a simulation code) for all element types, the element variable truth table is an optional mechanism for specifying whether a particular element result is output for the elements in a particular element block. For example, hydrostatic stress may be an output result for the elements in element block 3, but not those in element block 6.

It is helpful to describe the truth table as a two dimensional array, as shown in Table 4, “Element Variable Truth Table,” Each row of the array is associated with an element variable; each column of the array is associated with an element block. If a datum in the truth table is zero (`table(i, j)=0`), then no results are output for the  $i$ th element variable for the  $j$ th element block. A nonzero entry indicates that the appropriate result will be output. In this example, element variable 1 will be stored for all element blocks; element variable 2 will be stored for element blocks 1 and 4; and element variable 3 will be stored for element blocks 3 and 4. The table is stored such that the variable index cycles faster than the block index.

**Table 4** Element Variable Truth Table

	Elem Block #1	Elem Block #2	Elem Block #3	Elem Block #4
Elem Var #1	1	1	1	1
Elem Var #2	1	0	0	1
Elem Var #3	0	0	1	1

## 4 Application Programming Interface (API)

EXODUS II files can be written and read by application codes written in C, C++, or Fortran via calls to functions in the application programming interface (API). Functions within the API are categorized as data file utilities, model description functions, or results data functions.

In general, the following pattern is followed for writing data objects to a file:

1. create the file with `ex_create` (or `EXCRE` for Fortran);
2. write out global parameters to the file using `ex_put_init` (or `EXPINI` for Fortran);
3. write out specific data object parameters; for example, put out element block parameters with `ex_put_elem_block` (or `EXPELB` for Fortran);
4. write out the data object; for example, put out the connectivity for an element block with `ex_put_elem_conn` (or `EXPELC` for Fortran);
5. close the file with `ex_close` (or `EXCLOS` for Fortran).

Steps 3 and 4 are repeated within this pattern for each data object (i.e., nodal coordinates, element blocks, node sets, side sets, results variables, etc.). For some data object types, steps 3 and 4 are combined in a single call. For instance, `ex_put_qa` (or `EXPQA` for Fortran) writes out the parameters (number of QA records) as well as the data object itself (the QA records). During the database writing process, there are a few order dependencies (e.g., an element block must be written before element variables for that element block are written) which are documented in the description of each library function.

The invocation of the EXODUS II API functions for reading data is order independent, providing random read access. The following steps are typically used for reading data:

1. open the file with `ex_open` (or `EXOPEN` for Fortran);
2. read the global parameters for dimensioning purposes with `ex_get_init` (or `EXGINI` for Fortran);
3. read specific data object parameters; for example, read node set parameters with `ex_get_node_set_param` (or `EXGNSP` for Fortran);
4. read the data object; for example, read the node set node list with `ex_get_node_set` (or `EXGNS` for Fortran);
5. close the file with `ex_close` (or `EXCLOS` for Fortran).

Again, steps 3 and 4 are repeated for each object. For some object parameters, step 3 may be accomplished with a call to `ex_inquire` (or `EXINQ` for Fortran) to inquire the size of certain objects.

In developing applications using the EXODUS II API, the following points may prove beneficial:

- All functions that write objects to the database begin with `ex_put_` (`EXP` for Fortran); functions that read objects from the database begin with `ex_get_` (`EXG` for Fortran).
- Function arguments are classified as readable (`R`), writable (`W`), or both (`RW`). Readable arguments are not modified by the API routines; writable arguments are modified; read-write arguments may be either depending on the value of the argument.
- All application codes which use the EXODUS II API must include the file `'exodusII.h'` for C or `'exodusII.inc'` for Fortran. These files define constants that are used (1) as arguments to the API routines, (2) to set global parameters such as maximum string length and database version, and (3) as error condition or function return values.
- Throughout this section, sample code segments have been included to aid the application developer in using the API routines. These segments are not complete and there has been no attempt to include all calling sequence dependencies within them. Additionally, most arrays in the Fortran coding examples are shown dimensioned to some maximum value (i.e., `MAXQA`, `MAXINF`, `MAXNOD`, etc.). These values are not predefined constants so the library routines cannot check actual numbers of records against them. They are shown in this document simply to give an indication of how to statically dimension the arrays if necessary.
- Because 2-dimensional arrays cannot be statically dimensioned, either dynamic dimensioning or user indexing is required. Most of the sample code segments utilize user indexing within 1-dimensional arrays even though the variables are logically 2-dimensional.
- There are many netCDF utilities that prove useful. `ncdump`, which converts a binary netCDF file to a readable ASCII file, is the most notable.
- Because netCDF buffers I/O, it is important to flush all buffers (with `ex_update` in C or `EXUPDA` in Fortran) when debugging an application that produces an EXODUS II file.

## 4.1 Data File Utilities

This section describes data file utility functions for creating / opening a file, initializing a file with global parameters, reading / writing information text, inquiring on parameters stored in the data file, and error reporting .

## 4.1.1 Create EXODUS II File

The function `ex_create` or (`EXCRE` for Fortran) creates a new EXODUS II file and returns an ID that can subsequently be used to refer to the file.

All floating point values in an EXODUS II file are stored as either 4-byte (“float” in C; “REAL\*4” in FORTRAN) or 8-byte (“double” in C; “REAL\*8” or “DOUBLE PRECISION” in FORTRAN) numbers; no mixing of 4- and 8-byte numbers in a single file is allowed. An application code can compute either 4- or 8-byte values and can designate that the values be stored in the EXODUS II file as either 4- or 8-byte numbers; conversion between the 4- and 8-byte values is performed automatically by the API routines. Thus, there are four possible combinations of compute word size and storage (or I/O) word size.

In case of an error, `ex_create` returns a negative number; `EXCRE` returns a nonzero error number in `IERR`. Possible causes of errors include:

- Passing a file name that includes a directory that does not exist.
- Specifying a file name of a file that exists and also specifying a no clobber option.
- Attempting to create a file in a directory without permission to create files there.
- Passing an invalid file clobber mode.

### `ex_create`: C Interface

```
int ex_create (path, cmode, comp_ws, io_ws);
```

```
char* path (R)
```

The file name of the new EXODUS II file. This can be given as either an absolute path name (from the root of the file system) or a relative path name (from the current directory).

```
int cmode (R)
```

Clobber mode. Use one of the following predefined constants:

- `EX_NOCLOBBER` To create the new file only if the given file name does not refer to a file that already exists.
- `EX_CLOBBER` To create the new file, regardless of whether a file with the same name already exists. If a file with the same name does exist, its contents will be erased.

```
int* comp_ws (RW)
```

The word size in bytes (0, 4 or 8) of the floating point variables used in the application program. If 0 (zero) is passed, the default `sizeof(float)` will be used and returned in this variable. **WARNING:** all EXODUS II functions requiring floats must be passed floats declared with this passed in or returned compute word size (4 or 8).

```
int* io_ws (R)
```

The word size in bytes (4 or 8) of the floating point data as they are to be stored in the EXODUS II file.

The following code segment creates an EXODUS II file called `test.exo`:

```

#include "exodusII.h"
int CPU_word_size, IO_word_size, exoid;
CPU_word_size = sizeof(float);      /* use float or double */
IO_word_size = 8;                    /* store variables as doubles */
/* create EXODUS II file */
exoid = ex_create ("test.exo",      /* filename path */
                  EX_CLOBBER,       /* create mode */
                  &CPU_word_size,  /* CPU float word size in bytes */
                  &IO_word_size);  /* I/O float word size in bytes */

```

## EXCRE: Fortran Interface

INTEGER FUNCTION EXCRE (PATH, ICMODE, ICOMPWS, IOWS, IERR)

CHARACTER\*(\*) PATH (R)

The file name of the new EXODUS II file. This can be given as either an absolute path name (from the root of the file system) or a relative path name (from the current directory).

INTEGER ICMODE (R)

Clobber mode. Use one of the following predefined constants:

- EXNOCL        To create the new file only if the given file name does not refer to a file that already exists.
- EXCLOB        To create the new file, regardless of whether a file with the same name already exists. If a file with the same name does exist, its contents will be erased.

INTEGER ICOMPWS (RW)

The word size in bytes (0, 4 or 8) of the floating point (REAL) variables used in the application program. If 0 (zero) is passed, the default size of floating point values for the machine will be used and returned in this variable. **WARNING:** all EXODUS II functions requiring reals must be passed reals declared with this passed in or returned compute word size (4 or 8).

INTEGER IOWS (R)

The word size in bytes (4 or 8) of the floating point (REAL) data as they are to be stored in the EXODUS II file.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code segment creates an EXODUS II file called `test.exo`, specifying default values for compute and I/O word sizes:

```

      include 'exodusII.inc'
      integer cpu_ws, io_ws
c create EXODUS II files;
c REAL variables are default reals; store in file as DOUBLE PRECISION
      cpu_ws = 0
      io_ws = 8
      idexo = excre ('test.exo', EXCLOB, cpu_ws, io_ws, ierr)

```

## 4.1.2 Open EXODUS II File

The function `ex_open` or (`EXOPEN` for Fortran) opens an existing EXODUS II file and returns an ID that can subsequently be used to refer to the file, the word size of the floating point values stored in the file, and the version of the EXODUS II database (returned as a “float” in C or “REAL” in Fortran, regardless of the compute or I/O word size). Multiple files may be “open” simultaneously.

In case of an error, `ex_open` returns a negative number; `EXOPEN` returns a nonzero error number in `IERR`. Possible causes of errors include:

- The specified file does not exist.
- The mode specified is something other than the predefined constant `EX_READ` (`EXREAD` for Fortran) or `EX_WRITE` (`EXWRIT` for Fortran).
- Database version is earlier than 2.0.

### `ex_open`: C Interface

```
int ex_open (path, mode, comp_ws, io_ws, version);
```

`char* path (R)`

The file name of the EXODUS II file. This can be given as either an absolute path name (from the root of the file system) or a relative path name (from the current directory).

`int mode (R)`

Access mode. Use one of the following predefined constants:

- `EX_READ` To open the file just for reading.
- `EX_WRITE` To open the file for writing and reading.

`int* comp_ws (RW)`

The word size in bytes (0, 4 or 8) of the floating point variables used in the application program. If 0 (zero) is passed, the default size of floating point values for the machine will be used and returned in this variable. WARNING: all EXODUS II functions requiring reals must be passed reals declared with this passed in or returned compute word size (4 or 8).

`int* io_ws (RW)`

The word size in bytes (0, 4 or 8) of the floating point data as they are stored in the EXODUS II file. If the word size does not match the word size of data stored in the file, a fatal error is returned. If this argument is 0, the word size of the floating point data already stored in the file is returned.

`float* version (W)`

Returned EXODUS II database version number. The current version is 2.02

The following opens an EXODUS II file named `test.exo` for read only, using default settings for compute and I/O word sizes:

```
#include "exodusII.h"
int CPU_word_size,IO_word_size, exoid;
float version;
```

```

CPU_word_size = sizeof(float);      /* float or double */
IO_word_size = 0;                   /* use what is stored in file */

/* open EXODUS II files */
exoid = ex_open ("test.exo",        /* filename path */
                EX_READ,            /* access mode = READ */
                &CPU_word_size,    /* CPU word size */
                &IO_word_size,    /* IO word size */
                &version);         /* ExodusII library version */

```

## EXOPEN: Fortran Interface

```

INTEGER FUNCTION EXOPEN (PATH, IMODE, ICOMPWS, IOWS, VERS,
                        IERR)

```

```

CHARACTER*(*) PATH (R)

```

The file name of the EXODUS II file. This can be given as either an absolute path name (from the root of the file system) or a relative path name (from the current directory).

```

INTEGER IMODE (R)

```

Access mode. Use one of the following predefined constants:

- EXREAD            To open the file just for reading.
- EXWRIT           To open the file for writing and reading.

```

INTEGER ICOMPWS (RW)

```

The word size in bytes (0, 4 or 8) of the floating point variables used in the application program. If 0 (zero) is passed, the default size of floating point values for the machine will be used and returned in this variable. WARNING: all EXODUS II functions requiring reals must be passed reals declared with this passed in or returned compute word size.

```

INTEGER IOWS (RW)

```

The word size in bytes (0, 4 or 8) of the floating point data as they are stored in the EXODUS II file. If the word size does not match the word size of data stored in the file, a fatal error is returned. If this argument is 0, the word size of the floating point data already stored in the file is returned.

```

REAL VERS (W)

```

Returned EXODUS II version number. The current version is 2.02

```

INTEGER IERR (W)

```

Returned error code. If no errors occurred, 0 is returned.

The following opens an EXODUS II file named `test.exo` for read only, using default settings for compute and I/O word sizes:

```

      include 'exodusII.inc'
      integer cpu_ws, io_ws
      real vers
c
c open EXODUS II file
      cpu_ws = 0
      io_ws = 0
      idexo = exopen ('test.exo', EXREAD, cpu_ws, io_ws, vers, ierr)

```



### 4.1.3 Close EXODUS II File

The function `ex_close` or (`EXCLOS` for Fortran) updates and then closes an open EXODUS II file.

In case of an error, `ex_close` returns a negative number; a warning will return a positive number. `EXCLOS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).

#### **ex\_close: C Interface**

```
int ex_close (exoid);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

The following code segment closes an open EXODUS II file:

```
int error,exoid;
```

```
error = ex_close (exoid);
```

#### **EXCLOS: Fortran Interface**

```
SUBROUTINE EXCLOS ( IDEXO, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment closes an open EXODUS II file:

```
call exclos (idexo, ierr)
```

## 4.1.4 Update EXODUS II File

The function `ex_update` or (`EXUPDA` for Fortran) flushes all buffers to an EXODUS II file that is open for writing. This routine insures that the EXODUS II file is current.

In case of an error, `ex_update` returns a negative number; a warning will return a positive number. `EXUPDA` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).

### **ex\_update: C Interface**

```
int ex_update (exoid);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

The following code segment flushes all buffers to an open EXODUS II file:

```
int error,exoid;

error = ex_update (exoid);
```

### **EXUPDA: Fortran Interface**

```
SUBROUTINE EXUPDA ( IDEXO, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment flushes all buffers to an open EXODUS II file:

```
c
c update the data file; this should be done at the end of every
c time step to ensure that no data is lost if the analysis dies
c
      call exupda (idexo, ierr)
```

## 4.1.5 Write Initialization Parameters

The function `ex_put_init` (EXPINI in Fortran) writes the initialization parameters to the EXODUS II file. This function must be called once (and only once) before writing any data to the file.

In case of an error, `ex_put_init` returns a negative number; a warning will return a positive number. EXPINI returns a nonzero error (negative) or warning (positive) number in IERR. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (EXCRE or EXOPEN for Fortran).
- data file opened for read only.
- this routine has been called previously.

### `ex_put_init`: C Interface

```
int ex_put_init (exoid, title, num_dim, num_nodes, num_elem,  
                num_elem_blk, num_node_sets, num_side_sets);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char* title (R)`

Database title. Maximum length is MAX\_LINE\_LENGTH.

`int num_dim (R)`

The dimensionality of the database. This is the number of coordinates per node.

`int num_nodes (R)`

The number of nodal points.

`int num_elem (R)`

The number of elements.

`int num_elem_blk (R)`

The number of element blocks.

`int num_node_sets (R)`

The number of node sets.

`int num_side_sets (R)`

The number of side sets.

The following code segment will initialize an open EXODUS II file with the specified parameters:

```
int num_dim, num_nods, num_el, num_el_blk, num_ns, num_ss, error, exoid;  
  
/* initialize file with parameters */  
num_dim = 3; num_nods = 46; num_el = 5; num_el_blk = 5;  
num_ns = 2; num_ss = 5;  
error = ex_put_init (exoid, "This is the title", num_dim,  
                    num_nods, num_el, num_el_blk, num_ns, num_ss);
```

## EXPINI: Fortran Interface

```
SUBROUTINE EXPINI (IDEXO, TITLE, NDIM, NUMNP, NUMEL, NELBLK,  
                  NUMNPS, NUMESS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
CHARACTER*MXLNLN TITLE (R)
```

Database title.

```
INTEGER NDIM (R)
```

The dimensionality of the database. This is the number of coordinates per node.

```
INTEGER NUMNP (R)
```

The number of nodal points.

```
INTEGER NUMEL (R)
```

The number of elements.

```
INTEGER NELBLK (R)
```

The number of element blocks.

```
INTEGER NUMNPS (R)
```

The number of node sets.

```
INTEGER NUMESS (R)
```

The number of side sets.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will initialize an open EXODUS II file with the specified parameters:

```
        include 'exodusII.inc'  
        character*(MXLNLN) title  
c  
c initialize file with parameters  
c  
        title = "This is the title"  
        num_dim = 2  
        num_nodes = 8  
        num_elem = 2  
        num_elem_blk = 2  
        num_node_sets = 2  
        num_side_sets = 2  
  
        call expini (idexo, title, num_dim, num_nodes, num_elem,  
1         num_elem_blk, num_node_sets, num_side_sets, ierr)
```

## 4.1.6 Read Initialization Parameters

The function `ex_get_init` (EXGINI in Fortran) reads the initialization parameters from an opened EXODUS II file.

In case of an error, `ex_get_init` returns a negative number; a warning will return a positive number. EXGINI returns a nonzero error (negative) or warning (positive) number in IERR. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (EXCRE or EXOPEN for Fortran).

### `ex_get_init`: C Interface

```
int ex_get_init (exoid, title, num_dim, num_nodes, num_elem,  
                num_elem_blk, num_node_sets, num_side_sets);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char* title (W)`

Returned database title. String length may be up to `MAX_LINE_LENGTH` bytes.

`int* num_dim (W)`

Returned dimensionality of the database. This is the number of coordinates per node.

`int* num_nodes (W)`

Returned number of nodal points.

`int* num_elem (W)`

Returned number of elements.

`int* num_elem_blk (W)`

Returned number of element blocks.

`int* num_node_sets (W)`

Returned number of node sets.

`int* num_side_sets (W)`

Returned number of side sets.

The following code segment will read the initialization parameters from the open EXODUS II file:

```
#include "exodusII.h"  
  
int num_dim, num_nodes, num_elem, num_elem_blk,  
    num_node_sets, num_side_sets, error, exoid;  
char title[MAX_LINE_LENGTH+1];  
  
/* read database parameters */  
  
error = ex_get_init (exoid, title, &num_dim, &num_nodes,  
                    &num_elem, &num_elem_blk, &num_node_sets, &num_side_sets);
```

## EXGINI: Fortran Interface

```
SUBROUTINE EXGINI (IDEXO, TITLE, NDIM, NUMNP, NUMEL, NELBLK,  
                  NUMNPS, NUMESS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
CHARACTER*MXLNLN TITLE (W)
```

Returned database title.

```
INTEGER NDIM (W)
```

Returned dimensionality of the database. This is the number of coordinates per node.

```
INTEGER NUMNP (W)
```

Returned number of nodal points.

```
INTEGER NUMEL (W)
```

Returned number of elements.

```
INTEGER NELBLK (W)
```

Returned number of element blocks.

```
INTEGER NUMNPS (W)
```

Returned number of node sets.

```
INTEGER NUMESS (W)
```

Returned number of side sets.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will read the initialization parameters from the open EXODUS II file:

```
      character*(MXLNLN) titl  
c  
c read database parameters  
c  
      call exgini (idexo, titl, num_dim, num_nodes, num_elem,  
1      num_elem_blk, num_node_sets, num_side_sets, ierr)
```

## 4.1.7 Write QA Records

The function `ex_put_qa` (or `EXPQA` for Fortran) writes the QA records to the database. Each QA record contains four `MAX_STR_LENGTH`-byte character strings. The character strings are:

- 1) the analysis code name
- 2) the analysis code QA descriptor
- 3) the analysis date
- 4) the analysis time

In case of an error, `ex_put_qa` returns a negative number; a warning will return a positive number. `EXPQA` returns a nonzero error (negative) or warning (positive) number in `IERR`.

Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- QA records already exist in file.

### `ex_put_qa`: C Interface

```
int ex_put_qa (exoid, num_qa_records, qa_record[][4]);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int num_qa_records (R)`

The number of QA records.

`char* qa_record (R)`

Array containing the QA records.

The following code segment will write out two QA records:

```
int num_qa_rec, error, exoid;
char *qa_record[2][4];

/* write QA records */

num_qa_rec = 2;

qa_record[0][0] = "TESTWT1";
qa_record[0][1] = "testwt1";
qa_record[0][2] = "07/07/93";
qa_record[0][3] = "15:41:33";
qa_record[1][0] = "FASTQ";
qa_record[1][1] = "fastq";
qa_record[1][2] = "07/07/93";
qa_record[1][3] = "16:41:33";

error = ex_put_qa (exoid, num_qa_rec, qa_record);
```

## EXPQA: Fortran Interface

```
SUBROUTINE EXPQA (IDEXO, NQAREC, QAREC, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER NQAREC (R)
```

The number of QA records.

```
CHARACTER*MXSTLN QAREC (4,*) (R)
```

Array containing the QA records.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will write out two QA records:

```
c NOTE:      MAXQA is the maximum number of QA records
c
      include 'exodusII.inc'
      character*(MXSTLN) qa_record(4,MAXQA)
c
c write QA records
c
      num_qa_rec = 2

      qa_record(1,1) = "TESTWT2"
      qa_record(2,1) = "testwt2"
      qa_record(3,1) = "07/07/93"
      qa_record(4,1) = "15:41:33"
      qa_record(1,2) = "FASTQ"
      qa_record(2,2) = "fastq"
      qa_record(3,2) = "07/07/93"
      qa_record(4,2) = "16:41:33"

      call expqa (idexo, num_qa_rec, qa_record, ierr)
```



## 4.1.8 Read QA Records

The function `ex_get_qa` (or `EXGQA` for Fortran) reads the QA records from the database. Each QA record contains four `MAX_STR_LENGTH`-byte character strings. The character strings are:

- 1) the analysis code name
- 2) the analysis code QA descriptor
- 3) the analysis date
- 4) the analysis time

Memory must be allocated for the QA records before this call is made. The number of QA records can be determined by invoking `ex_inquire` (or `EXINQ` in Fortran). See Section 4.1.11 on page 41.

In case of an error, `ex_get_qa` returns a negative number; a warning will return a positive number. `EXGQA` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no QA records were stored.

### `ex_get_qa`: C Interface

```
int ex_get_qa (exoid, qa_record[][4]);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char* qa_record (W)`

Returned array containing the QA records.

The following will determine the number of QA records and read them from the open EXODUS II file:

```
#include "exodusII.h"

int num_qa_rec, error, exoid
char *qa_record[MAX_QA_REC][4];

/* read QA records */

ex_inquire (exoid, EX_INQ_QA, &num_qa_rec, &fdum, cdum);

for (i=0; i<num_qa_rec; i++)
  for (j=0; j<4; j++)
    qa_record[i][j] =
      (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));

error = ex_get_qa (exoid, qa_record);
```

## EXGQA: Fortran Interface

```
SUBROUTINE EXGQA (IDEXO, QAREC, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
CHARACTER*MXSTLN QAREC(4,*) (W)
```

Returned array containing the QA records.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following will determine the number of QA records and read them from the open EXODUS II file:

```
C NOTE:      MAXQA is the maximum number of QA records
C
      include 'exodusII.inc'
      character*(MXSTLN) qa_record(4,MAXQA)
C
C read QA records
C
      call exinq (idexo, EXQA, num_qa_rec, fdum, cdum, ierr)

      call exgqa (idexo, qa_record, ierr)
```

## 4.1.9 Write Information Records

The function `ex_put_info` (or `EXPINF` for Fortran) writes information records to the database. The records are `MAX_LINE_LENGTH`-character strings.

In case of an error, `ex_put_info` returns a negative number; a warning will return a positive number. `EXPINF` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- information records already exist in file.

### `ex_put_info`: C Interface

```
int ex_put_info (exoid, num_info, info);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int num_info (R)
```

The number of information records.

```
char** info (R)
```

Array containing the information records.

The following code will write out three information records to an open EXODUS II file:

```
int error, exoid, num_info;
char *info[3];

/* write information records */

num_info = 3;

info[0] = "This is the first information record.";
info[1] = "This is the second information record.";
info[2] = "This is the third information record.";

error = ex_put_info (exoid, num_info, info);
```

### `EXPINF`: Fortran Interface

```
SUBROUTINE EXPINF (IDEXO, NINFO, INFO, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER NINFO (R)

The number of information records.

CHARACTER\*MXLNLN INFO(\*) (R)

Array containing the information records.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code will write out three information records to an open EXODUS II file:

```
c NOTE:      MAXINF is the maximum number of information records
c
      include 'exodusII.inc'
      character*(MXLNLN) inform(MAXINF)
c
c write information records
c
      num_info = 3
      inform(1) = "This is the first information record."
      inform(2) = "This is the second information record."
      inform(3) = "This is the third information record."
      call expinf (idexo, num_info, inform, ierr)
```

## 4.1.10 Read Information Records

The function `ex_get_info` (or `EXGINF` for Fortran) reads information records from the database. The records are `MAX_LINE_LENGTH`-character strings. Memory must be allocated for the information records before this call is made. The number of records can be determined by invoking `ex_inquire` (or `EXINQ` in Fortran). See Section 4.1.11.

In case of an error, `ex_get_info` returns a negative number; a warning will return a positive number. `EXGINF` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no information records were stored.

### `ex_get_info`: C Interface

```
int ex_get_info (exoid, info);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char** info (W)`

Returned array containing the information records.

The following code segment will determine the number of information records and read them from an open EXODUS II file:

```
#include "exodusII.h"

int error, exoid, num_info;
char *info[MAXINFO];

/* read information records */

error = ex_inquire (exoid, EX_INQ_INFO, &num_info, &fdum, &cdum);

for (i=0; i<num_info; i++)
    info[i] = (char *) calloc ((MAX_LINE_LENGTH+1), sizeof(char));

error = ex_get_info (exoid, info);
```

### `EXGINF`: Fortran Interface

```
SUBROUTINE EXGINF (IDEXO, INFO, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`CHARACTER*MXLNLN INFO(*) (W)`

Returned array containing the information records.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code segment will determine the number of information records and read them from an open EXODUS II file:

```
c NOTE:      MAXINF is the maximum number of information records
c
      include 'exodusII.inc'
      character*(MXLNLN) inform(MAXINF)
c
c read information records
c
      call exinq (idexo, EXINFO, num_info, fdum, cdum, ierr)
      call exginf (idexo, inform, ierr)
```

## 4.1.11 Inquire EXODUS Parameters

The function `ex_inquire` (or `EXINQ` in Fortran) is used to inquire values of certain data entities in an EXODUS II file. Memory must be allocated for the returned values before this function is invoked.

In case of an error, `ex_inquire` returns a negative number; a warning will return a positive number. `EXINQ` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- requested information not stored in the file.
- invalid request flag.

### `ex_inquire`: C Interface

```
int ex_inquire (exoid, req_info, ret_int, ret_float,  
               ret_char);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int req_info (R)
```

A flag which designates what information is requested. It must be one of the following constants (predefined in the file `exodusII.h`):

- `EX_INQ_API_VERS` The EXODUS II API version number is returned in `ret_float`. The API version number reflects the release of the function library (i.e., function names, argument list, etc.). The current API version is 2.03.
- `EX_INQ_DB_VERS` The EXODUS II database version number is returned in `ret_float`. The database version number reflects the format of the data in the EXODUS II file. The current database version is 2.02.
- `EX_INQ_TITLE` The title stored in the database is returned in `ret_char`.
- `EX_INQ_DIM` The dimensionality, or number of coordinates per node (1, 2 or 3), of the database is returned in `ret_int`.
- `EX_INQ_NODES` The number of nodal points is returned in `ret_int`.
- `EX_INQ_ELEM` The number of elements is returned in `ret_int`.
- `EX_INQ_ELEM_BLK` The number of element blocks is returned in `ret_int`.
- `EX_INQ_NODE_SETS` The number of node sets is returned in `ret_int`.
- `EX_INQ_NS_NODE_LEN` The length of the concatenated node sets node list is returned in `ret_int`.
- `EX_INQ_NS_DF_LEN` The length of the concatenated node sets distribution list is returned in `ret_int`.
- `EX_INQ_SIDE_SETS` The number of side sets is returned in `ret_int`.
- `EX_INQ_SS_ELEM_LEN` The length of the concatenated side sets element list is returned in `ret_int`.

- EX\_INQ\_SS\_DF\_LEN The length of the concatenated side sets distribution factor list is returned in `ret_int`.
- EX\_INQ\_SS\_NODE\_LEN The aggregate length of all of the side sets node lists is returned in `ret_int`.
- EX\_INQ\_EB\_PROP The number of integer properties stored for each element block is returned in `ret_int`; this number includes the property named "ID".
- EX\_INQ\_NS\_PROP The number of integer properties stored for each node set is returned in `ret_int`; this number includes the property named "ID".
- EX\_INQ\_SS\_PROP The number of integer properties stored for each side set is returned in `ret_int`; this number includes the property named "ID".
- EX\_INQ\_QA The number of QA records is returned in `ret_int`.
- EX\_INQ\_INFO The number of information records is returned in `ret_int`.
- EX\_INQ\_TIME The number of time steps stored in the database is returned in `ret_int`.

`int* ret_int (W)`

Returned integer, if an integer value is requested (according to `req_info`); otherwise, supply a dummy argument.

`float* ret_float (W)`

Returned float, if a float value is requested (according to `req_info`); otherwise, supply a dummy argument.

`char* ret_char (W)`

Returned single character, if a character value is requested (according to `req_info`); otherwise, supply a dummy argument.

As an example, the following will return the number of element block properties stored in the EXODUS II file:

```
#include "exodusII.h"
int error, exoid, num_props;
float fdum;
char *cdum;

/* determine the number of element block properties */

error = ex_inquire (exoid, EX_INQ_EB_PROP, &num_props, &fdum, cdum);
```

## EXINQ: Fortran Interface

```
SUBROUTINE EXINQ (IDEXO, INFREQ, INTRET, RELRET, CHRRET,
  IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.



INTEGER INFREQ (R)

A flag which designates what information is requested. It must be one of the following constants (predefined in the file `exodusII.inc`):

- EXVERS           The EXODUS II API version number is returned in RELRET. The API version number reflects the release of the function library (i.e., function names, argument list, etc.). The current API version is 2.03.
- EXDBVR           The EXODUS II database version number is returned in RELRET. The database version number reflects the format of the data in the EXODUS II file. The current database version is 2.02.
- EXTITL           The title stored in the database is returned in CHRRET.
- EXDIM            The dimensionality, or number of coordinates per node (1, 2 or 3), of the database is returned in INTRET.
- EXNODE           The number of nodal points is returned in INTRET.
- EXELEM           The number of elements is returned in INTRET.
- EXELBL           The number of element blocks is returned in INTRET.
- EXNODS           The number of node sets is returned in INTRET.
- EXNSNL           The length of the concatenated node sets node list is returned in INTRET.
- EXNSDF           The length of the concatenated node sets distribution factors list is returned in INTRET.
- EXSIDS           The number of side sets is returned in INTRET.
- EXSSEL           The length of the concatenated side sets element list is returned in INTRET.
- EXSSDF           The length of the concatenated side sets distribution factors list is returned in INTRET.
- EXSSNL           The aggregate length of all of the side sets node lists is returned in INTRET.
- EXNEBP           The number of integer properties stored for each element block is returned in INTRET; this number includes the property named "ID".
- EXNNSP           The number of integer properties stored for each node set is returned in INTRET; this number includes the property named "ID".
- EXNSSP           The number of integer properties stored for each side set is returned in INTRET; this number includes the property named "ID".
- EXQA             The number of QA records is returned in INTRET.
- EXINFO           The number of information records is returned in INTRET.
- EXTIMS            The number of time steps stored in the database is returned in INTRET.

INTEGER INTRET (W)

Returned integer, if an integer value is requested (according to INFREQ); otherwise, supply a dummy argument.

REAL RELRET (W)

Returned float, if a float value is requested (according to INFREQ); otherwise, supply a dummy argument.

CHARACTER\*(\*) CHRRET (W)

Returned single character, if a character value is requested (according to INFREQ); otherwise, supply a dummy argument.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

As an example, the following will return the number of element block properties stored in the EXODUS II file:

```
        include 'exodusII.inc'

        real fdum
        character*1 cdum
c
c read element block properties
c
        call exinq (idexo, EXNEBP, num_props, fdum, cdum, ierr)
```

## 4.1.12 Error Reporting

The function `ex_err` or (`EXERR` for Fortran) logs an error to `stderr`. It is intended to provide explanatory messages for error codes returned from other EXODUS II routines. This function does not return an error code.

The passed in error codes and corresponding messages are listed in Appendix C. The programmer may supplement the error message printed for standard errors by providing an error message. If the error code is provided with no error message, the predefined message will be used. The error code `EX_MSG` is available to log application specific messages.

### `ex_err`: C Interface

```
void ex_err (module_name, message, err_num);
```

`char* module_name (R)`

This is a string containing the name of the calling function.

`char* message (R)`

This is a string containing a message explaining the error or problem. If `EX_VERBOSE` (see `ex_opts`) is true, this message will be printed to `stderr`. Otherwise, nothing will be printed.

`int err_num (R)`

This is an integer code identifying the error. EXODUS II C functions place an error code value in `exrrval`, an external `int`. Negative values are considered fatal errors while positive values are warnings. There is a set of predefined values defined in `exodusII.h`. The predefined constant `EX_PRTLASTMSG` will cause the last error message to be output, regardless of the setting of the error reporting level (see `ex_opts`).

The following is an example of the use of this function:

```
#include "exodusII.h"
int exoid, CPU_word_size, IO_word_size, errval;
float version;
char errmsg[MAX_ERR_LENGTH];

CPU_word_size = sizeof(float);
IO_word_size = 0;

/* open EXODUS II file */

if (exoid = ex_open ("test.exo", EX_READ, &CPU_word_size, &IO_word_size,
    &version)
    {
    errval = 999;
    sprintf(errmsg, "Error: cannot open file test.exo");
    ex_err("prog_name", errmsg, errval);
    }
```

## EXERR: Fortran Interface

```
SUBROUTINE EXERR (MODNAM, MSG, ERRNUM)
```

```
CHARACTER*MXSTLN MODNAM (R)
```

This is a string containing the name of the calling function.

```
CHARACTER*MXLNLN MSG (R)
```

This is a string containing a message explaining the error or problem. If EXVRBS (see EXOPTS) is true, this message will be printed to `stderr`. Otherwise, nothing will be printed.

```
INTEGER ERRNUM (R)
```

This is an integer code identifying the error. EXODUS II Fortran functions place an error code value in `ierr`, a returned value. Negative values are considered fatal errors while positive values are warnings. There is a set of predefined values defined in `exodusII.inc`. The predefined constant `PRTMSG` will cause the last error message to be output, regardless of the setting of the error reporting level (see EXOPTS)

The following is an example of the use of this function:

```
        include 'exodusII.inc'

        integer cpu_ws
c
c open EXODUS II files
c

        cpu_ws = 0
        io_ws = 0

        idexo = exopen ("test.exo", EXREAD, cpu_ws, io_ws, vers, ierr)

        if (ierr .lt. 0) then
c
c     error was fatal, so print it out; override setting of exopts
c
            call exerr ("progname", "", PRTMSG)
        endif
```

## 4.1.13 Set Error Reporting Level

The function `ex_opts` (or `EXOPTS` for Fortran) is used to set message reporting options.

In case of an error, `ex_opts` returns a negative number; a warning will return a positive number. `EXOPTS` returns a nonzero error (negative) or warning (positive) number in `IERR`.

### **ex\_opts: C Interface**

```
int ex_opts (option_val);
```

```
int option_val (R)
```

Integer option value. Current options are:

- `EX_ABORT` Causes fatal errors to force program exit. (Default is false.)
- `EX_DEBUG` Causes certain messages to print for debug use. (Default is false.)
- `EX_VERBOSE` Causes all error messages to print when true, otherwise no error messages will print. (Default is false.)

NOTE: Values may be OR'ed together to provide any combination of these capabilities.

For example, the following will cause all messages to print and will cause the program to exit upon receipt of fatal error:

```
#include "exodusII.h"

ex_opts (EX_ABORT | EX_VERBOSE);
```

### **EXOPTS: Fortran Interface**

```
SUBROUTINE EXOPTS (OPTVAL, IERR)
```

```
INTEGER OPTVAL (R)
```

Integer option value. Current options are:

- `EXABRT` Causes fatal errors to force program exit. (Default is false.)
- `EXDEBG` Causes certain messages to print for debug use. (Default is false.)
- `EXVRBS` Causes all error messages to print when true, otherwise no error messages will print. (Default is false.)

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

NOTE: Values may be OR'ed together to provide any combination of capabilities.

For example, the following will cause all messages to print:

```
include 'exodusII.inc'
call exopts (EXVRBS, IERR)
```

## **4.2 Model Description**

The routines in this section read and write information which describe an EXODUS II finite element model. This includes nodal coordinates, element order map, element connectivity arrays, element attributes, node sets, side sets, and object properties.

## 4.2.1 Write Nodal Coordinates

The function `ex_put_coord` (or `EXPCOR` for Fortran) writes the coordinates of the nodes in the model. The function `ex_put_init` (`EXPINI` for Fortran) must be invoked before this call is made.

Because the coordinates are floating point values, the application code must declare the arrays passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_coord` returns a negative number; a warning will return a positive number. `EXPCOR` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).

### `ex_put_coord`: C Interface

```
int ex_put_coord (exoid, x_coor, y_coor, z_coor);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
void* x_coor (R)
```

The X coordinates of the nodes.

```
void* y_coor (R)
```

The Y coordinates of the nodes. These are stored only if `num_dim > 1`; otherwise, pass in dummy address.

```
void* z_coor (R)
```

The Z coordinates of the nodes. These are stored only if `num_dim > 2`; otherwise, pass in dummy address.

The following will write the nodal coordinates to an open EXODUS II file:

```
int error, exoid;
```

```
/* if file opened with compute word size of sizeof(float) */  
float x[8], y[8], z[8];
```

```
/* write nodal coordinates values to database */
```

```
x[0] = 0.0; y[0] = 0.0; z[0] = 0.0;  
x[1] = 0.0; y[1] = 0.0; z[1] = 1.0;  
x[2] = 1.0; y[2] = 0.0; z[2] = 1.0;  
x[3] = 1.0; y[3] = 0.0; z[3] = 0.0;
```

```

x[4] = 0.0; y[4] = 1.0; z[4] = 0.0;
x[5] = 0.0; y[5] = 1.0; z[5] = 1.0;
x[6] = 1.0; y[6] = 1.0; z[6] = 1.0;
x[7] = 1.0; y[7] = 1.0; z[7] = 0.0;

error = ex_put_coord (exoid, x, y, z);

```

## EXPCOR: Fortran Interface

```

SUBROUTINE EXPCOR (IDEXO, XN, YN, ZN, IERR)

```

```

INTEGER IDEXO (R)

```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```

REAL XN(*) (R)

```

The X coordinates of the nodes.

```

REAL YN(*) (R)

```

The Y coordinates of the nodes. These are stored only if NDIM > 1; otherwise, pass in a dummy address.

```

REAL ZN(*) (R)

```

The Z coordinates of the nodes. These are stored only if NDIM > 2; otherwise, pass in a dummy address.

```

INTEGER IERR (W)

```

Returned error code. If no errors occurred, 0 is returned.

The following will write the nodal coordinates to an open EXODUS II file:

```

      real x(8), y(8), dummy(1)
c
c write nodal coordinates values for a 2-d model to the database
c

```

```

x(1) = 0.0
x(2) = 1.0
x(3) = 1.0
x(4) = 0.0
x(5) = 1.0
x(6) = 2.0
x(7) = 2.0
x(8) = 1.0
y(1) = 0.0
y(2) = 0.0
y(3) = 1.0
y(4) = 1.0
y(5) = 0.0
y(6) = 0.0
y(7) = 1.0
y(8) = 1.0

```

```

call expcor (idexo, x, y, dummy, ierr)

```



## 4.2.2 Read Nodal Coordinates

The function `ex_get_coord` or (`EXGCOR` for Fortran) reads the coordinates of the nodes. Memory must be allocated for the coordinate arrays (`x_coor`, `y_coor`, and `z_coor`) before this call is made. The length of each of these arrays is the number of nodes in the mesh.

Because the coordinates are floating point values, the application code must declare the arrays passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_coord` returns a negative number; a warning will return a positive number. `EXGCOR` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if nodal coordinates were not stored.

### `ex_get_coord`: C Interface

```
int ex_get_coord (exoid, x_coor, y_coor, z_coor);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
void* x_coor (W)
```

Returned X coordinates of the nodes.

```
void* y_coor (W)
```

Returned Y coordinates of the nodes. These are returned only if `num_dim > 1`; otherwise, pass in a dummy address.

```
void* z_coor (W)
```

Returned Z coordinates of the nodes. These are returned only if `num_dim > 2`; otherwise, pass in a dummy address.

The following code segment will read the nodal coordinates from an open EXODUS II file:

```
int error, exoid;
float *x, *y, *z;

/* read nodal coordinates values from database */

x = (float *) calloc(num_nodes, sizeof(float));
y = (float *) calloc(num_nodes, sizeof(float));
if (num_dim >= 3)
    z = (float *) calloc(num_nodes, sizeof(float));
else
    z = 0;
error = ex_get_coord (exoid, x, y, z);
```

## EXGCOR: Fortran Interface

```
SUBROUTINE EXGCOR (IDEXO, XN, YN, ZN, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
REAL XN(*) (W)
```

Returned X coordinates of the nodes.

```
REAL YN(*) (W)
```

Returned Y coordinates of the nodes. These are returned only if NDIM > 1; otherwise, pass in a dummy address.

```
REAL ZN(*) (W)
```

Returned Z coordinates of the nodes. These are returned only if NDIM > 2; otherwise, pass in a dummy address.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will read the nodal coordinates from an open EXODUS II file:

```
c NOTE:      MAXNOD is the maximum number of nodes
c
c      real x(MAXNOD), y(MAXNOD), z(MAXNOD)
c
c read nodal coordinates values from database
c
c
c      call exgcor (idexo, x, y, z, ierr)
```

### 4.2.3 Write Coordinate Names

The function `ex_put_coord_names` or (`EXPCON` for Fortran) writes the names of the coordinate arrays to the database. The function `ex_put_init` (`EXPINI` for Fortran) must be invoked before this call is made.

In case of an error, `ex_put_coord_names` returns a negative number; a warning will return a positive number. `EXPCON` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).

#### `ex_put_coord_names`: C Interface

```
int ex_put_coord_names (exoid, coord_names);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char** coord_names (R)`  
Array containing `num_dim` names (of length `MAX_STR_LENGTH`) of the nodal coordinate arrays.

The following coding will write the coordinate names to an open EXODUS II file:

```
int error, exoid;
char *coord_names[3];

coord_names[0] = "xcoor";
coord_names[1] = "ycoor";
coord_names[2] = "zcoor";

error = ex_put_coord_names (exoid, coord_names);
```

#### `EXPCON`: Fortran Interface

```
SUBROUTINE EXPCON (IDEXO, NAMECO, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`CHARACTER*MXSTLN NAMECO(*) (R)`  
Array containing `NDIM` names for the nodal coordinate arrays.

`INTEGER IERR (W)`  
Returned error code. If no errors occurred, 0 is returned.

The following coding will write the coordinate names to an open EXODUS II file:

```
include 'exodusII.inc'  
character*(MXSTLN)coord_names(3)  
  
coord_names(1) = "xcoor"  
coord_names(2) = "ycoor"  
coord_names(3) = "zcoor"  
  
call expcon (idexo, coord_names, ierr)
```

## 4.2.4 Read Coordinate Names

The function `ex_get_coord_names` or (`EXGCON` for Fortran) reads the names (`MAX_STR_LENGTH`-characters in length) of the coordinate arrays from the database. Memory must be allocated for the character strings before this function is invoked.

In case of an error, `ex_get_coord_names` returns a negative number; a warning will return a positive number. `EXGCON` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if coordinate names were not stored.

### `ex_get_coord_names`: C Interface

```
int ex_get_coord_names (exoid, coord_names);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

char\*\* coord\_names (W)

Returned pointer to a vector containing `num_dim` names of the nodal coordinate arrays.

The following code segment will read the coordinate names from an open EXODUS II file:

```
int error, exoid;
char *coord_names[3];

for (i=0; i<num_dim; i++)
    coord_names[i] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));

error = ex_get_coord_names (exoid, coord_names);
```

### `EXGCON`: Fortran Interface

```
SUBROUTINE EXGCON (IDEXO, NAMECO, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

CHARACTER\*MXSTLN NAMECO(\*) (W)

Returned array containing `NDIM` names for the nodal coordinate arrays.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code segment will read the coordinate names from an open EXODUS II file:

```
character*(MXSTLN) coord_names(3)

call exgcon (idexo, coord_names, ierr)
```

## 4.2.5 Write Node Number Map

The function `ex_put_node_num_map` (or `EXPNNM` for Fortran) writes out the optional node number map to the database. The function `ex_put_init` (`EXPINI` for Fortran) must be invoked before this call is made.

In case of an error, `ex_put_node_num_map` returns a negative number; a warning will return a positive number. `EXPNNM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- a node number map already exists in the file.

### `ex_put_node_num_map`: C Interface

```
int ex_put_node_num_map (exoid, node_map);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_map (R)`  
The node number map.

The following code generates a default node number map and outputs it to an open EXODUS II file. This is a trivial case and included just for illustration. Since this map is optional, it should be written out only if it contains something other than the default map.

```
int *node_map, error, exoid;
```

```
node_map = (int *) calloc(num_nodes, sizeof(int));
```

```
for (i=1; i<=num_nodes; i++)  
    node_map[i-1] = i;
```

```
error = ex_put_node_num_map (exoid, node_map);
```

### `EXPNNM`: Fortran Interface

```
SUBROUTINE EXPNNM (IDEXO, MAPNOD, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER MAPNOD(*) (R)`  
The node number map.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code generates a default node number map and outputs it to an open EXODUS II file. This is a trivial case and included just for illustration. Since this map is optional, it should be written out only if it contains something other than the default map.

```
c NOTE:      MAXNOD is the maximum number of nodes
c
      integer node_map(MAXNOD)
c
c write node order map
c
      do 10 i = 1, num_nodes
          node_map(i) = i
10    continue
      call expnrm (idexo, node_map, ierr)
```

## 4.2.6 Read Node Number Map

The function `ex_get_node_num_map` (or `EXGNNM` for Fortran) reads the optional node number map from the database. If a node number map is not stored in the data file, a default array (1,2,3, . . . `num_nodes`) is returned. Memory must be allocated for the node number map array (`num_nodes` in length) before this call is made.

In case of an error, `ex_get_node_num_map` returns a negative number; a warning will return a positive number. `EXGNNM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- if a node number map is not stored, a default map and a warning value are returned.

### `ex_get_node_num_map`: C Interface

```
int ex_get_node_num_map (exoid, node_map);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_map (W)`

Returned node number map.

The following code will read a node number map from an open EXODUS II file:

```
int *node_map, error, exoid;

/* read node number map */
node_map = (int *) calloc(num_nodes, sizeof(int));
error = ex_get_node_num_map (exoid, node_map);
```

### EXGNNM: Fortran Interface

```
SUBROUTINE EXGNNM (IDEXO, MAPNOD, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER MAPNOD(*) (W)`

Returned node number map.

`INTEGER IERR (W)`

Returned error code. If no errors occurred, 0 is returned.

The following code will read a node number map from an open EXODUS II file:

```
integer node_map(MAXNODES)
c
c read node number map
call exgnnm (idexo, node_map, ierr)
```



## 4.2.7 Write Element Number Map

The function `ex_put_elem_num_map` (or `EXPENM` for Fortran) writes out the optional element number map to the database. The function `ex_put_init` (`EXPINI` for Fortran) must be invoked before this call is made.

In case of an error, `ex_put_elem_num_map` returns a negative number; a warning will return a positive number. `EXPENM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- an element number map already exists in the file.

### `ex_put_elem_num_map`: C Interface

```
int ex_put_elem_num_map (exoid, elem_map);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int* elem_map (R)
```

The element number map.

The following code generates a default element number map and outputs it to an open EXODUS II file. This is a trivial case and included just for illustration. Since this map is optional, it should be written out only if it contains something other than the default map.

```
int *elem_map, error, exoid;

elem_map = (int *) calloc(num_elem, sizeof(int));

for (i=1; i<=num_elem; i++)
    elem_map[i-1] = i;

error = ex_put_elem_num_map (exoid, elem_map);
```

### **EXPENM: Fortran Interface**

```
SUBROUTINE EXPENM (IDEXO, MAPEL, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER MAPEL(*) (R)
```

The element number map.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code generates a default element number map and outputs it to an open EXODUS II file. This is a trivial case and included just for illustration. Since this map is optional, it should be written out only if it contains something other than the default map.

```
c NOTE:      MAXELEM is the maximum number of elements
c
c      integer elem_map(MAXELEM)
c
c write element number map
c
c
c      do 10 i = 1, num_elem
c          elem_map(i) = i
10  continue
c
c      call expenm (idexo, elem_map, ierr)
```

## 4.2.8 Read Element Number Map

The function `ex_get_elem_num_map` (or `EXGENM` for Fortran) reads the optional element number map from the database. If an element number map is not stored in the data file, a default array (1,2,3,...`num_elem`) is returned. Memory must be allocated for the element number map array (`num_elem` in length) before this call is made.

In case of an error, `ex_get_elem_num_map` returns a negative number; a warning will return a positive number. `EXGENM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- if an element number map is not stored, a default map and a warning value are returned.

### `ex_get_elem_num_map`: C Interface

```
int ex_get_elem_num_map (exoid, elem_map);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* elem_map (W)`

Returned element number map.

The following code will read an element number map from an open EXODUS II file:

```
int *elem_map, error, exoid;

/* read element number map */
elem_map = (int *) calloc(num_elem, sizeof(int));
error = ex_get_elem_num_map (exoid, elem_map);
```

### EXGENM: Fortran Interface

```
SUBROUTINE EXGENM (IDEXO, MAPEL, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER MAPEL(*) (W)`

Returned element number map.

`INTEGER IERR (W)`

Returned error code. If no errors occurred, 0 is returned.

The following code will read an element number map from an open EXODUS II file:

```
integer elem_map(MAXELEM)
c
c read element number map
call exgenm (idexo, elem_map, ierr)
```

## 4.2.9 Write Element Order Map

The function `ex_put_map` (or `EXPMAP` for Fortran) writes out the optional element order map to the database. The function `ex_put_init` (`EXPINI` for Fortran) must be invoked before this call is made.

In case of an error, `ex_put_map` returns a negative number; a warning will return a positive number. `EXPMAP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- an element map already exists in the file.

### `ex_put_map`: C Interface

```
int ex_put_map (exoid, elem_map);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int* elem_map (R)
```

The element order map.

The following code generates a default element order map and outputs it to an open EXODUS II file. This is a trivial case and included just for illustration. Since this map is optional, it should be written out only if it contains something other than the default map.

```
int *elem_map, error, exoid;

elem_map = (int *) calloc(num_elem, sizeof(int));

for (i=1; i<=num_elem; i++)
    elem_map[i-1] = i;

error = ex_put_map (exoid, elem_map);
```

### `EXPMAP`: Fortran Interface

```
SUBROUTINE EXPMAP (IDEXO, MAPEL, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER MAPEL(*) (R)
```

The element order map.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code generates a default element order map and outputs it to an open EXODUS II file. This is a trivial case and included just for illustration. Since this map is optional, it should be written out only if it contains something other than the default map.

```
c NOTE:      MAXELEM is the maximum number of elements
c
      integer elem_map(MAXELEM)
c
c write element order map
c
      do 10 i = 1, num_elem
          elem_map(i) = i
10    continue

      call expmap (idexo, elem_map, ierr)
```

## 4.2.10 Read Element Order Map

The function `ex_get_map` (or `EXGMAP` for Fortran) reads the element order map from the database. If an element order map is not stored in the data file, a default array (1,2,3, . . . `num_elem`) is returned. Memory must be allocated for the element map array (`num_elem` in length) before this call is made.

In case of an error, `ex_get_map` returns a negative number; a warning will return a positive number. `EXGMAP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- if an element order map is not stored, a default map and a warning value are returned.

### `ex_get_map`: C Interface

```
int ex_get_map (exoid, elem_map);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* elem_map (W)`

Returned element order map.

The following code will read an element order map from an open EXODUS II file:

```
int *elem_map, error, exoid;

/* read element order map */
elem_map = (int *) calloc(num_elem, sizeof(int));
error = ex_get_map (exoid, elem_map);
```

### `EXGMAP`: Fortran Interface

```
SUBROUTINE EXGMAP (IDEXO, MAPEL, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER MAPEL(*) (W)`

Returned element order map.

`INTEGER IERR (W)`

Returned error code. If no errors occurred, 0 is returned.

The following code will read an element order map from an open EXODUS II file:

```
integer elem_map(MAXELEM)
c
c read element order map
call exgmap (idexo, elem_map, ierr)
```

## 4.2.11 Write Element Block Parameters

The function `ex_put_elem_block` (or `EXPELB` for Fortran) writes the parameters used to describe an element block.

In case of an error, `ex_put_elem_block` returns a negative number; a warning will return a positive number. `EXPELB` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- an element block with the same ID has already been specified.
- the number of element blocks specified in the call to `ex_put_init` (`EXPINI` for Fortran) has been exceeded.

### `ex_put_elem_block`: C Interface

```
int ex_put_elem_block (exoid, elem_blk_id, elem_type,
                      num_elem_this_blk, num_nodes_per_elem, num_attr);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id (R)`

The element block ID.

`char* elem_type (R)`

The type of elements in the element block. The maximum length of this string is `MAX_STR_LENGTH`. For historical reasons, this string should be all upper case.

`int num_elem_this_blk (R)`

The number of elements in the element block.

`int num_nodes_per_elem (R)`

The number of nodes per element in the element block.

`int num_attr (R)`

The number of attributes per element in the element block.

For example, the following code segment will initialize an element block with an ID of 10, write out the connectivity array, and write out the element attributes array:

```
int id, error, exoid, num_elem_in_blk, num_nodes_per_elem,
    *connect, num_attr;
float *attrib;

/* write element block parameters */

id = 10;
num_elem_in_blk = 2;
```

```

num_nodes_per_elem = 4;          /* elements are 4-node shells */
num_attr = 1;                   /* one attribute per element */

error = ex_put_elem_block (exoid, id, "SHEL",
    num_elem_in_blk, num_nodes_per_elem, num_attr);

/* write element connectivity */

connect = (int *)
    calloc (num_elem_in_blk*num_nodes_per_elem, sizeof(int));

/* fill connect with node numbers; nodes for first element*/
connect[0] = 1; connect[1] = 2; connect[2] = 3; connect[3] = 4;
/* nodes for second element */
connect[4] = 5; connect[5] = 6; connect[6] = 7; connect[7] = 8;

error = ex_put_elem_conn (exoid, id, connect);

/* write element block attributes */

attrib = (float *) calloc (num_attr * num_elem_in_blk, sizeof(float));

for (i=0, cnt=0; i<num_elem_in_blk; i++)
    for (j=0; j<num_attr; j++, cnt++)
        attrib[cnt] = 1.0;

error = ex_put_elem_attr (exoid, id, attrib);

```

## EXPELB: Fortran Interface

```

SUBROUTINE EXPELB (IDEXO, IDELB, NAMELB, NUMELB, NUMLNK,
    NUMATR, IERR)

```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IDELB (R)

The element block ID.

CHARACTER\*MXSTLN NAMELB (R)

The type of elements in the element block. For historical reasons, this string should be all upper case.

INTEGER NUMELB (R)

The number of elements in the element block.

INTEGER NUMLNK (R)

The number of nodes per element in the element block.

INTEGER NUMATR (R)

The number of attributes per element in the element block.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.



For example, the following code segment will initialize an element block with an ID of 10, write out the connectivity array, and write out the element attributes array:

```
c NOTE:      MAXLNK is the maximum number of nodes per element
c           MAXELB is the maximum number of elements per element block
c           MAXATR is the maximum number of attributes per element
c
c           include 'exodusII.inc'
c
c           integer ebid, connect(MAXLNK * MAXELB)
c           real attrib(MAXATR * MAXELB)
c           character*(MXSTLN) cname
c
c write element block parameters
c
c           ebid = 10
c           cname = "SHEL"
c           numelb = 2
c           numlnk = 4
c           numatr = 1
c
c           call expelb (idexo, ebid, cname, numelb, numlnk, numatr, ierr)
c
c fill element connectivity and write it out;
c nodes for first element
c           connect(1) = 1
c           connect(2) = 2
c           connect(3) = 3
c           connect(4) = 4
c
c nodes for second element
c           connect(5) = 5
c           connect(6) = 6
c           connect(7) = 7
c           connect(8) = 8
c
c           call expelc (idexo, ebid, connect, ierr)
c
c write element block attributes
c
c           icnt = 0
c           do 20 i=1,numelb
c             do 10 j=1,numatr
c               icnt = icnt + 1
c               attrib(icnt) = 1.0
10          continue
20        continue
c
c           call expeat (idexo, ebid, attrib, ierr)
```

## 4.2.12 Read Element Block Parameters

The function `ex_get_elem_block` (or `EXGELB` for Fortran) reads the parameters used to describe an element block. IDs of all element blocks stored can be determined by calling `ex_get_elem_blk_ids` (`EXGEBI` for Fortran).

In case of an error, `ex_get_elem_block` returns a negative number; a warning will return a positive number. `EXGELB` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- element block with specified ID is not stored in the data file.

### `ex_get_elem_block`: C Interface

```
int ex_get_elem_block (exoid, elem_blk_id, elem_type,
                      num_elem_this_blk, num_nodes_per_elem, num_attr);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id (R)`

The element block ID.

`char* elem_type (W)`

Returned type of elements in the element block. The maximum length of this string is `MAX_STR_LENGTH`.

`int* num_elem_this_blk (W)`

Returned number of elements in the element block.

`int* num_nodes_per_elem (W)`

Returned number of nodes per element in the element block.

`int* num_attr (W)`

Returned number of attributes per element in the element block.

As an example, the following code segment will read the parameters for the element block with an ID of 10 and read the connectivity and element attributes arrays from an open EXODUS II file:

```
#include "exodusII.h"
int id, error, exoid, num_el_in_blk, num_nod_per_el, num_attr, *connect;
float *attrib;
char elem_type[MAX_STR_LENGTH+1];

/* read element block parameters */
id = 10;
error = ex_get_elem_block (exoid, id, elem_type,
                          &num_el_in_blk, &num_nod_per_elem, &num_attr);

/* read element connectivity */
```

```

connect = (int *) calloc(num_nod_per_el*num_el_in_blk, sizeof(int));
error = ex_get_elem_conn (exoid, id, connect);
/* read element block attributes */
attrib = (float *) calloc (num_attr * num_el_in_blk, sizeof(float));
error = ex_get_elem_attr (exoid, id, attrib);

```

## EXGELB: Fortran Interface

```

SUBROUTINE EXGELB (IDEXO, IDELB, NAMELB, NUMELB, NUMLNK,
                  NUMATR, IERR)

```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IDELB (R)

The element block ID.

CHARACTER\*MXSTLN NAMELB (W)

The type of elements in the element block.

INTEGER NUMELB (W)

Returned number of elements in the element block.

INTEGER NUMLNK (W)

Returned number of nodes per element in the element block.

INTEGER NUMATR (W)

Returned number of attributes per element in the element block.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

As an example, the following code segment will read the parameters for the element block with an ID of 10 and the connectivity and element attributes arrays associated with that element block:

```

c NOTE:      MAXLNK is the maximum number of nodes per element
c           MAXELB is the maximum number of elements per element block
c           MAXATR is the maximum number of attributes per element

      include 'exodusII.inc'

      integer connect(MAXLNK * MAXELB)
      real attrib(MAXATR * MAXELB)
      character*(MXSTLN) typ
c
c read element block parameters
      id = 10
      call exgelb (idexo, id, typ, numelb, numlnk, numatt, ierr)
c
c read element connectivity
      call exgelc (idexo, id, connect, ierr)
c
c read element block attributes
      call exgeat (idexo, id, attrib, ierr)

```

## 4.2.13 Read Element Blocks IDs

The function `ex_get_elem_blk_ids` (or `EXGEBI` for Fortran) reads the IDs of all of the element blocks. Memory must be allocated for the returned array of (`num_elem_blk`) IDs before this function is invoked. The required size (`num_elem_blk`) can be determined via a call to `ex_inquire` (or `EXINQ` for Fortran).

In case of an error, `ex_get_elem_blk_ids` returns a negative number; a warning will return a positive number. `EXGEBI` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).

### `ex_get_elem_blk_ids`: C Interface

```
int ex_get_elem_blk_ids (exoid, elem_blk_ids);
```

`int exoid` (R)  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* elem_blk_ids` (W)  
Returned array of the element blocks IDs. The order of the IDs in this array reflects the sequence that the element blocks were introduced into the file.

The following code segment reads all the element block IDs:

```
int error, exoid, *idelbs, num_elem_blk;  
  
idelbs = (int *) calloc(num_elem_blk, sizeof(int));  
  
error = ex_get_elem_blk_ids (exoid, idelbs);
```

### `EXGEBI`: Fortran Interface

```
SUBROUTINE EXGEBI (IDEXO, IDELBS, IERR)
```

`INTEGER IDEXO` (R)  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDELBS(*)` (W)  
Returned array of element blocks IDs. The order of the IDs in this array reflects the sequence that the element blocks were introduced into the file.

`INTEGER IERR` (W)  
Returned error code. If no errors occurred, 0 is returned.

The following code segment reads all the element block IDs:

```
c NOTE:      MAXEBL is the maximum number of element blocks  
c  
      integer idelbs(MAXEBL)  
      call exgebi (idexo, idelbs, ierr)
```

## 4.2.14 Write Element Block Connectivity

The function `ex_put_elem_conn` (or `EXPELC` for Fortran) writes the connectivity array for an element block. The function `ex_put_elem_block` (`EXPELB` for Fortran) must be invoked before this call is made.

In case of an error, `ex_put_elem_conn` returns a negative number; a warning will return a positive number. `EXPELC` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_elem_block` was not called previously.

### `ex_put_elem_conn`: C Interface

```
int ex_put_elem_conn (exoid, elem_blk_id, connect);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id (R)`  
The element block ID.

`int connect[num_elem_this_blk,num_nodes_per_elem] (R)`  
The connectivity array; a list of nodes (internal node IDs; see Section 3.5 on page 7) that define each element in the element block. The node index cycles faster than the element index.

Refer to the description of `ex_put_elem_block` (`EXPELB` for Fortran) for an example of a code segment that writes out the connectivity array for an element block.

### **EXPELC: Fortran Interface**

```
SUBROUTINE EXPELC (IDEXO, IDELB, LINK, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDELB (R)`  
The element block ID.

`INTEGER LINK(NUMLNK,NUMELB) (R)`  
The connectivity array; a list of nodes (internal node IDs; see Section 3.5 on page 7) that define each element. The node index cycles faster than the element index.

`INTEGER IERR (W)`  
Returned error code. If no errors occurred, 0 is returned.

Refer to the description of `ex_put_elem_block` (`EXPELB` for Fortran) for an example of a code segment that writes out the connectivity array for an element block.

## 4.2.15 Read Element Block Connectivity

The function `ex_get_elem_conn` (or `EXGELC` for Fortran) reads the connectivity array for an element block. Memory must be allocated for the connectivity array (`num_elem_this_blk * num_nodes_per_elem` in length) before this routine is called.

In case of an error, `ex_get_elem_conn` returns a negative number; a warning will return a positive number. **EXGELC** returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- an element block with the specified ID is not stored in the file.

### `ex_get_elem_conn`: C Interface

```
int ex_get_elem_conn (exoid, elem_blk_id, connect);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id (R)`  
The element block ID.

`int connect[num_elem_this_blk,num_nodes_per_elem] (W)`  
Returned connectivity array; a list of nodes (internal node IDs; see Section 3.5 on page 7) that define each element. The node index cycles faster than the element index.

For an example of a code segment that reads the connectivity for an element block, refer to the description of `ex_get_elem_block`.

### **EXGELC: Fortran Interface**

```
SUBROUTINE EXGELC (IDEXO, IDELB, LINK, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDELB (R)`  
The element block ID.

`INTEGER LINK (NUMLNK, NUMELB) (W)`  
Returned connectivity array; a list of nodes (internal node IDs; see Section 3.5 on page 7) that define each element. The node index cycles faster than the element index.

`INTEGER IERR (W)`  
Returned error code. If no errors occurred, 0 is returned.

For an example of a code segment that reads the connectivity for an element block, refer to the description of `EXGELB`.

## 4.2.16 Write Element Block Attributes

The function `ex_put_elem_attr` (or `EXPEAT` for Fortran) writes the attributes for an element block. Each element in the element block must have the same number of attributes, so there are  $(\text{num\_attr} * \text{num\_elem\_this\_blk})$  attributes for each element block. The function `ex_put_elem_block` (`EXPELB` for Fortran) must be invoked before this call is made.

Because the attributes are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_elem_attr` returns a negative number; a warning will return a positive number. `EXPEAT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_elem_block` was not called previously for specified element block ID.
- `ex_put_elem_block` was called with 0 attributes specified.

### `ex_put_elem_attr`: C Interface

```
int ex_put_elem_attr (exoid, elem_blk_id, attrib);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int elem_blk_id (R)
```

The element block ID.

```
void attrib[num_elem_this_blk,num_attr] (R)
```

The list of attributes for the element block. The `num_attr` index cycles faster.

Refer to the description of `ex_put_elem_block` for an example of a code segment that writes out the attributes array for an element block.

### `EXPEAT`: Fortran Interface

```
SUBROUTINE EXPEAT (IDEXO, IDELB, ATRIB, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER IDELB (R)
```

The element block ID.

REAL ATTRIB(NUMATR,NUMELB) (R)

The list of attributes for the element block. The NUMATR index cycles faster.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

Refer to the description of EXPELB for an example of a code segment that writes out the attributes array for an element block.



## 4.2.17 Read Element Block Attributes

The function `ex_get_elem_attr` (or `EXGEAT` for Fortran) reads the attributes for an element block. Memory must be allocated for `(num_attr * num_elem_this_blk)` attributes before this routine is called.

Because the attributes are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_elem_attr` returns a negative number; a warning will return a positive number. `EXGEAT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- invalid element block ID.
- a warning value is returned if no attributes are stored in the file.

### **ex\_get\_elem\_attr: C Interface**

```
int ex_get_elem_attr (exoid, elem_blk_id, attrib);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id (R)`  
The element block ID.

`void attrib[num_elem_this_blk,num_attr] (W)`  
Returned list of `(num_attr * num_elem_this_blk)` attributes for the element block, with the `num_attr` index cycling faster.

For an example of a code segment that reads the element attributes for an element block, refer to the description of `ex_get_elem_block`.

### **EXGEAT: Fortran Interface**

```
SUBROUTINE EXGEAT (IDEXO, IDELB, ATRIB, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDELB (R)`  
The element block ID.

`REAL ATRIB(NUMATR,NUMELB) (W)`  
Returned list of `(NUMATR*NUMELB)` attributes for the element block, with the `NUMATR` index cycling faster.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of a code segment that reads the element attributes for an element block, refer to the description of EXGELB.

## 4.2.18 Write Node Set Parameters

The function `ex_put_node_set_param` (or `EXPNP` for Fortran) writes the node set ID, the number of nodes which describe a single node set, and the number of distribution factors for the node set.

In case of an error, `ex_put_node_set_param` returns a negative number; a warning will return a positive number. `EXPNP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- the number of node sets specified in the call to `ex_put_init` (`EXPINI` for Fortran) was zero or has been exceeded.
- a node set with the same ID has already been stored.
- the specified number of distribution factors is not zero and is not equal to the number of nodes.

### `ex_put_node_set_param`: C Interface

```
int ex_put_node_set_param (exoid, node_set_id,  
                          num_nodes_in_set, num_dist_in_set);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id (R)`

The node set ID.

`int num_nodes_in_set (R)`

The number of nodes in the node set.

`int num_dist_in_set (R)`

The number of distribution factors in the node set. This should be either 0 (zero) for no factors, or should equal `num_nodes_in_set`.

The following code segment will write out a node set to an open EXODUS II file:

```
int id, num_nodes_in_set, num_dist_in_set, error, exoid, *node_list;  
float *dist_fact;  
  
/* write node set parameters */  
  
id = 20; num_nodes_in_set = 5; num_dist_in_set = 5;  
error = ex_put_node_set_param (exoid, id, num_nodes_in_set,  
                              num_dist_in_set);
```

```

/* write node set node list */
node_list = (int *) calloc (num_nodes_in_set, sizeof(int));
node_list[0] = 100; node_list[1] = 101; node_list[2] = 102;
node_list[3] = 103; node_list[4] = 104;
error = ex_put_node_set (exoid, id, node_list);

/* write node set distribution factors */

dist_fact = (float *) calloc (num_dist_in_set, sizeof(float));
dist_fact[0] = 1.0; dist_fact[1] = 2.0; dist_fact[2] = 3.0;
dist_fact[3] = 4.0; dist_fact[4] = 5.0;
error = ex_put_node_set_dist_fact (exoid, id, dist_fact);

```

## EXPNP: Fortran Interface

```
SUBROUTINE EXPNP (IDEXO, IDNPS, NNNPS, NDNPS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER IDNPS (R)
```

The node set ID.

```
INTEGER NNNPS (R)
```

The number of nodes in the node set.

```
INTEGER NDNPS (R)
```

The number of distribution factors in the node set. This should be either 0 (zero) for no factors, or should equal NNNPS.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will write out a node set to an open EXODUS II file:

```

integer node_list(5)
real dist_fact(5)
c
c write a single node set
c
call expnp (idexo, 20, 4, 4, ierr)
node_list(1) = 100
node_list(2) = 101
node_list(3) = 102
node_list(4) = 103

dist_fact(1) = 1.0
dist_fact(2) = 2.0
dist_fact(3) = 3.0
dist_fact(4) = 4.0

call expns (idexo, 20, node_list, ierr)
call expnsd (idexo, 20, dist_fact, ierr)

```

## 4.2.19 Read Node Set Parameters

The function `ex_get_node_set_param` (or `EXGNP` for Fortran) reads the number of nodes which describe a single node set and the number of distribution factors for the node set.

In case of an error, `ex_get_node_set_param` returns a negative number; a warning will return a positive number. `EXGNP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no node sets are stored in the file.
- incorrect node set ID.

### `ex_get_node_set_param`: C Interface

```
int ex_get_node_set_param (exoid, node_set_id,  
                           num_nodes_in_set, num_dist_in_set);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id (R)`

The node set ID.

`int* num_nodes_in_set (W)`

Returned number of nodes in the node set.

`int* num_dist_in_set (W)`

Returned number of distribution factors in the node set.

The following code segment will read a node set from an open EXODUS II file:

```
int error, exoid, id, num_nodes_in_set, num_df_in_set, *node_list;  
float *dist_fact;  
  
/* read node set parameters */  
  
id = 100;  
error = ex_get_node_set_param (exoid, id, &num_nodes_in_set,  
                               &num_df_in_set);  
  
/* read node set node list */  
  
node_list = (int *) calloc(num_nodes_in_set, sizeof(int));  
error = ex_get_node_set (exoid, id, node_list);  
  
/* read node set distribution factors */  
  
if (num_df_in_set > 0) {  
    dist_fact = (float *) calloc(num_nodes_in_set, sizeof(float));  
    error = ex_get_node_set_dist_fact (exoid, id, dist_fact); }  
}
```

## EXGNP: Fortran Interface

```
SUBROUTINE EXGNP (IDEXO, IDNPS, NNNPS, NDNPS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER IDNPS (R)
```

The node set ID.

```
INTEGER NNNPS (W)
```

Returned number of nodes in the node set.

```
INTEGER NDNPS (W)
```

Returned number of distribution factors in the node set.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will read all node sets from an open EXODUS II file:

```
c NOTE:      MAXNS is the maximum number of node sets
c           MAXNOD is the maximum number of nodes in a node set
c
c           integer ids(MAXNS), node_list(MAXNOD)
c           real dist_fact(MAXNOD)
c
c read individual node sets
c
c           if (num_node_sets .gt. 0) then
c             call exgnsi (idexo, ids, ierr)
c           endif
c
c           do 100 i = 1, num_node_sets
c             call exgnp (idexo, ids(i), nnnps, numdf, ierr)
c             call exgns (idexo, ids(i), node_list, ierr)
c             call exgnsd (idexo, ids(i), dist_fact, ierr)
100          continue
```

## 4.2.20 Write Node Set

The function `ex_put_node_set` (or `EXPNS` for Fortran) writes the node list for a single node set. The function `ex_put_node_set_param` (or `EXPNP` for Fortran) must be called before this routine is invoked.

In case of an error, `ex_put_node_set` returns a negative number; a warning will return a positive number. `EXPNS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_node_set_param` (or `EXPNP` for Fortran) not called previously.

### `ex_put_node_set`: C Interface

```
int ex_put_node_set (exoid, node_set_id,  
                    node_set_node_list);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id (R)`

The node set ID.

`int* node_set_node_list (R)`

Array containing the node list for the node set. Internal node IDs are used in this list (see Section 3.5 on page 7).

Refer to the description of `ex_put_node_set_param` for a sample code segment to write out a node set.

### `EXPNS`: Fortran Interface

```
SUBROUTINE EXPNS (IDEXO, IDNPS, LTNNPS, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDNPS (R)`

The node set ID.

`INTEGER LTNNPS(*) (R)`

Array containing the node list for the node set. Internal node IDs are used in this list (see Section 3.5 on page 7).

`INTEGER IERR (W)`

Returned error code. If no errors occurred, 0 is returned.

Refer to the description of `EXPNP` for a sample code segment to write out a node set.

## 4.2.21 Read Node Set

The function `ex_get_node_set` (or `EXGNS` for Fortran) reads the node list for a single node set. Memory must be allocated for the node list (`num_nodes_in_set` in length) before this function is invoked.

In case of an error, `ex_get_node_set` returns a negative number; a warning will return a positive number. `EXGNS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no node sets are stored in the file.
- incorrect node set ID.

### `ex_get_node_set`: C Interface

```
int ex_get_node_set (exoid, node_set_id,  
                    node_set_node_list);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id (R)`

The node set ID.

`int* node_set_node_list (W)`

Returned array containing the node list for the node set. Internal node IDs are used in this list (see Section 3.5 on page 7).

Refer to the description of `ex_get_node_set_param` for a sample code segment to read a node set.

### EXGNS: Fortran Interface

```
SUBROUTINE EXGNS (IDEXO, IDNPS, LTNNPS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER IDNPS (R)

The node set ID.

INTEGER LTNNPS(\*) (W)

Returned array containing the node list for the node set. Internal node IDs are used in this list (see Section 3.5 on page 7).

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

Refer to the description of `EXGNP` for a sample code segment to read a node set.



## 4.2.22 Write Node Set Distribution Factors

The function `ex_put_node_set_dist_fact` (or `EXPNSD` for Fortran) writes distribution factors for a single node set. The function `ex_put_node_set_param` (or `EXPNP` for Fortran) must be called before this routine is invoked.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_node_set_dist_fact` returns a negative number; a warning will return a positive number. `EXPNSD` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_node_set_param` (or `EXPNP` for Fortran) not called previously.
- a call to `ex_put_node_set_param` (or `EXPNP` for Fortran) specified zero distribution factors.

### `ex_put_node_set_dist_fact`: C Interface

```
int ex_put_node_set_dist_fact (exoid, node_set_id,  
                             node_set_dist_fact);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int node\_set\_id (R)

The node set ID.

void\* node\_set\_dist\_fact (R)

Array containing the distribution factors in the node set.

Refer to the description of `ex_put_node_set_param` for a sample code segment to write out the distribution factors for a node set.

### `EXPNSD`: Fortran Interface

```
SUBROUTINE EXPNSD (IDEXO, IDNPS, FACNPS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER IDNPS (R)

The node set ID.

REAL FACNPS( \* ) (R)

Array containing the distribution factors in the node set.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

Refer to the description of EXPNP for a sample code segment to write out the distribution factors for a node set.

## 4.2.23 Read Node Set Distribution Factors

The function `ex_get_node_set_dist_fact` (or `EXGNSD` for Fortran) returns the distribution factors for a single node set. Memory must be allocated for the list of distribution factors (`num_dist_in_set` in length) before this function is invoked.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_node_set_dist_fact` returns a negative number; a warning will return a positive number. `EXGNSD` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- a warning value is returned if no distribution factors were stored.

### `ex_get_node_set_dist_fact`: C Interface

```
int ex_get_node_set_dist_fact (exoid, node_set_id,  
                               node_set_dist_fact);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id (R)`  
The node set ID.

`void* node_set_dist_fact (W)`  
Returned array containing the distribution factors in the node set.

Refer to the description of `ex_get_node_set_param` for a sample code segment to read a node set’s distribution factors.

### `EXGNSD`: Fortran Interface

```
SUBROUTINE EXGNSD (IDEXO, IDNPS, FACNPS, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDNPS (R)`  
The node set ID.

`REAL FACNPS(*) (W)`  
Returned array containing the distribution factors in the node set.

`INTEGER IERR (W)`  
Returned error code. If no errors occurred, 0 is returned.

Refer to the description of `EXGNP` for a sample code segment to read a node set’s distribution factors.

## 4.2.24 Read Node Sets IDs

The function `ex_get_node_set_ids` (or `EXGNSI` for Fortran) reads the IDs of all of the node sets. Memory must be allocated for the returned array of `(num_node_sets)` IDs before this function is invoked.

In case of an error, `ex_get_node_set_ids` returns a negative number; a warning will return a positive number. `EXGNSI` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no node sets are stored in the file.

### `ex_get_node_set_ids`: C Interface

```
int ex_get_node_set_ids (exoid, node_set_ids);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int* node_set_ids (W)
```

Returned array of the node sets IDs. The order of the IDs in this array reflects the sequence the node sets were introduced into the file.

As an example, the following code will read all of the node set IDs from an open data file:

```
int *ids, num_node_sets, error, exoid;
/* read node sets IDs */
ids = (int *) calloc(num_node_sets, sizeof(int));
error = ex_get_node_set_ids (exoid, ids);
```

### `EXGNSI`: Fortran Interface

```
SUBROUTINE EXGNSI (IDEXO, IDNPSS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER IDNPSS(*) (W)
```

Returned array of node sets IDs. The order of the IDs in this array reflects the sequence the node sets were introduced into the file.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

As an example, the following code will read all of the node set IDs from an open EXODUS II file:

```
integer ids(MAXNS)
if (num_node_sets .gt. 0) then
    call exgnsi (idexo, ids, ierr)
endif
```

## 4.2.25 Write Concatenated Node Sets

The function `ex_put_concat_node_sets` (or `EXPCNS` for Fortran) writes the node set ID's, node sets node count array, node sets distribution factor count array, node sets node list pointers array, node sets distribution factor pointer, node set node list, and node set distribution factors for all of the node sets. "Concatenated node sets" refers to the arrays required to define all of the node sets (ID array, counts arrays, pointers arrays, node list array, and distribution factors array) as described in Section 3.10 on page 11. Writing concatenated node sets is more efficient than writing individual node sets. See Appendix A for a discussion of efficiency issues.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type ("float" or "double" in C; "REAL\*4" or "REAL\*8" in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_concat_node_sets` returns a negative number; a warning will return a positive number. `EXPCNS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- the number of node sets specified in a call to `ex_put_init` (`EXPINI` for Fortran) was zero or has been exceeded.
- a node set with the same ID has already been stored.
- the number of distribution factors specified for one of the node sets is not zero and is not equal to the number of nodes in the same node set.

### `ex_put_concat_node_sets`: C Interface

```
int ex_put_concat_node_sets (exoid, node_set_ids,  
    num_nodes_per_set, num_dist_per_set,  
    node_sets_node_index, node_sets_dist_index,  
    node_sets_node_list, node_sets_dist_fact);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_set_ids (R)`

Array containing the node set ID for each set.

`int* num_nodes_per_set (R)`

Array containing the number of nodes for each set.

`int* num_dist_per_set (R)`

Array containing the number of distribution factors for each set.

`int* node_sets_node_index (R)`  
 Array containing the indices into the `node_set_node_list` which are the locations of the first node for each set. These indices are 0-based.

`int* node_sets_dist_index (R)`  
 Array containing the indices into the `node_set_dist_list` which are the locations of the first distribution factor for each set. These indices are 0-based.

`int* node_sets_node_list (R)`  
 Array containing the nodes for all sets. Internal node IDs are used in this list (see Section 3.5 on page 7).

`void* node_sets_dist_fact (R)`  
 Array containing the distribution factors for all sets.

For example, the following code will write out two node sets in a concatenated format:

```
int ids[2], num_nodes_per_set[2], node_ind[2], node_list[8],
    num_df_per_set[2], df_ind[2], error, exoid;
float dist_fact[8];

ids[0] = 20; ids[1] = 21;

num_nodes_per_set[0] = 5; num_nodes_per_set[1] = 3;

node_ind[0] = 0; node_ind[1] = 5;

node_list[0] = 100; node_list[1] = 101; node_list[2] = 102;
node_list[3] = 103; node_list[4] = 104;
node_list[5] = 200; node_list[6] = 201; node_list[7] = 202;

num_df_per_set[0] = 5; num_df_per_set[1] = 3;

df_ind[0] = 0; df_ind[1] = 5;

dist_fact[0] = 1.0; dist_fact[1] = 2.0; dist_fact[2] = 3.0;
dist_fact[3] = 4.0; dist_fact[4] = 5.0;
dist_fact[5] = 1.1; dist_fact[6] = 2.1; dist_fact[7] = 3.1;

error = ex_put_concat_node_sets (exoid, ids, num_nodes_per_set,
    num_df_per_set, node_ind, df_ind, node_list, dist_fact);
```

## EXPCNS: Fortran Interface

```
SUBROUTINE EXPCNS (IDEXO, IDNPSS, NNNPS, NDNPS, IXNNPS,
    IXDNPS, LTNNPS, FACNPS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IDNPSS(\*) (R)

Array containing the node set ID for each set.

INTEGER NNNPS( \*) (R)

Array containing the number of nodes for each set.

INTEGER NDNPS( \*) (R)

Array containing the number of distribution factors for each set.

INTEGER IXNNPS( \*) (R)

Array containing the indices into the LTTNPS array which are the locations of the first node for each set. These indices are 1-based.

INTEGER IXDNPS( \*) (R)

Array containing the indices into the FACNPS array which are the locations of the first distribution factor for each set. These indices are 1-based.

INTEGER LTTNPS( \*) (R)

Array containing the nodes for all sets. Internal node IDs are used in this list (see Section 3.5 on page 7).

REAL FACNPS( \*) (R)

Array containing the distribution factors for all sets.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For example, the following code writes out two node sets in a concatenated format:

```
integer ids(2), nnnps(2), ndnps(2), nodeind(2), factind(2)
integer nodelist(8), distfact(8)

ids(1) = 20
ids(2) = 21

nnnps(1) = 5
nnnps(2) = 3

ndnps(1) = 5
ndnps(2) = 3

nodeind(1) = 1
nodeind(2) = 6

factind(1) = 1
factind(2) = 6

nodelist(1) = 100
nodelist(2) = 101
nodelist(3) = 102
nodelist(4) = 103
nodelist(5) = 104
nodelist(6) = 200
nodelist(7) = 201
nodelist(8) = 202
```

```
distfact(1) = 1.0  
distfact(2) = 2.0  
distfact(3) = 3.0  
distfact(4) = 4.0  
distfact(5) = 5.0  
distfact(6) = 1.1  
distfact(7) = 2.1  
distfact(8) = 3.1
```

```
call expcns (idexo, ids, nmnps, ndnps, nodeind, factind, nodelist,  
1 distfact, ierr)
```



## 4.2.26 Read Concatenated Node Sets

The function `ex_get_concat_node_sets` (or `EXGCNS` for Fortran) reads the node set ID's, node set node count array, node set distribution factors count array, node set node pointers array, node set distribution factors pointer array, node set node list, and node set distribution factors for all of the node sets. "Concatenated node sets" refers to the arrays required to define all of the node sets (ID array, counts arrays, pointers arrays, node list array, and distribution factors array) as described in Section 3.10 on page 11.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type ("float" or "double" in C; "REAL\*4" or "REAL\*8" in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

The length of each of the returned arrays can be determined by invoking `ex_inquire` (or `EXINQ` for Fortran). See Section 4.1.11 on page 41.

In case of an error, `ex_get_concat_node_sets` returns a negative number; a warning will return a positive number. `EXGCNS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no node sets are stored in the file.

### `ex_get_concat_node_sets`: C Interface

```
int ex_get_concat_node_sets (exoid, node_set_ids,  
    num_nodes_per_set, num_dist_per_set,  
    node_sets_node_index, node_sets_dist_index,  
    node_sets_node_list, node_sets_dist_fact);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_set_ids (W)`

Returned array containing the node set ID for each set.

`int* num_nodes_per_set (W)`

Returned array containing the number of nodes for each set.

`int* num_dist_per_set (W)`

Returned array containing the number of distribution factors for each set.

`int* node_sets_node_index (W)`

Returned array containing the indices into the `node_set_node_list` which are the locations of the first node for each set. These indices are 0-based.

`int* node_sets_dist_index (W)`

Returned array containing the indices into the `node_set_dist_fact` which are the locations of the first distribution factor for each set. These indices are 0-based.

`int* node_sets_node_list (W)`  
Returned array containing the nodes for all sets. Internal node IDs are used in this list (see Section 3.5 on page 7).

`void* node_sets_dist_fact (W)`  
Returned array containing the distribution factors for all sets.

As an example, the following code segment will read concatenated node sets:

```
#include "exodusII.h"

int error, exoid, num_node_sets, list_len, *ids, *num_nodes_per_set,
    *num_df_per_set, *node_ind, *df_ind, *node_list;
float *dist_fact

/* read concatenated node sets */

error = ex_inquire (exoid, EX_INQ_NODE_SETS, &num_node_sets, &fdum,
    cdum);

ids = (int *) calloc(num_node_sets, sizeof(int));
num_nodes_per_set = (int *) calloc(num_node_sets, sizeof(int));
num_df_per_set = (int *) calloc(num_node_sets, sizeof(int));
node_ind = (int *) calloc(num_node_sets, sizeof(int));
df_ind = (int *) calloc(num_node_sets, sizeof(int));

error = ex_inquire (exoid, EX_INQ_NS_NODE_LEN, &list_len, &fdum, cdum);
node_list = (int *) calloc(list_len, sizeof(int));

error = ex_inquire (exoid, EX_INQ_NS_DF_LEN, &list_len, &fdum, cdum);

dist_fact = (float *) calloc(list_len, sizeof(float));

error = ex_get_concat_node_sets (exoid, ids, num_nodes_per_set,
    num_df_per_set, node_ind, df_ind, node_list, dist_fact);
```

## **EXGCNS: Fortran Interface**

```
SUBROUTINE EXGCNS (IDEXO, IDNPSS, NNNPS, NDNPS, IXNNPS,
    IXDNPS, LTNNPS, FACNPS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IDNPSS(\*) (W)

Returned array containing the node set ID for each set.

INTEGER NNNPS(\*) (W)

Returned array containing the number of nodes for each set.

INTEGER NDNPS(\*) (W)

Returned array containing the number of distribution factors for each set.

INTEGER IXNNPS(\*) (W)

Returned array containing the indices into the LTNNPS array which are the locations of the first node for each set. These indices are 1-based.

INTEGER IXDNPS(\*) (W)

Returned array containing the indices into the FACNPS array which are the locations of the first distribution factor for each set. These indices are 1-based.

INTEGER LTNNPS(\*) (W)

Returned array containing the nodes for all sets. Internal node IDs are used in this list (see Section 3.5 on page 7).

REAL FACNPS(\*) (W)

Returned array containing the distribution factors for all sets.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

As an example, the following code segment will read concatenated node sets:

```
c NOTE:      MAXNS is the maximum number of node sets
c           MAXNOD is the maximum number of nodes in a node set
c
c           integer ids(MAXNS), numnodes(MAXNS), num_df(MAXNS), node_ind(MAXNS),
1           df_ind(MAXNS), node_list(MAXNOD*MAXNS), dist_fact(MAXNOD*MAXNS)
c
c read concatenated node sets
c
c           call exinq (idexo, EXNODS, num_node_sets, fdum, cdum, ierr)
c
c           if (num_node_sets .gt. 0) then
c
c use the next calls if you can dynamically allocate arrays
c
c           call exinq (idexo, EXNSNL, list_len, fdum, cdum, ierr)
c           call exinq (idexo, EXNSDF, list_len, fdum, cdum, ierr)
c
c           call exgcns (idexo, ids, numnodes, num_df,
1           node_ind, df_ind, node_list, dist_fact, ierr)
c           endif
```

## 4.2.27 Write Side Set Parameters

The function `ex_put_side_set_param` (or `EXPSP` for Fortran) writes the side set ID and the number of sides (faces on 3-d element types; edges on 2-d element types) which describe a single side set, and the number of distribution factors on the side set. Because each side of a side set is completely defined by an element and a local side number, the number of sides is equal to the number of elements in a side set.

In case of an error, `ex_put_side_set_param` returns a negative number; a warning will return a positive number. `EXPSP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- the number of side sets specified in the call to `ex_put_init` (`EXPINI` for Fortran) was zero or has been exceeded.
- a side set with the same ID has already been stored.

### `ex_put_side_set_param`: C Interface

```
int ex_put_side_set_param (exoid, side_set_id,  
                           num_side_in_set, num_dist_fact_in_set);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int side\_set\_id (R)

The side set ID.

int num\_side\_in\_set (R)

The number of sides (faces or edges) in the side set.

int num\_dist\_fact\_in\_set (R)

The number of distribution factors on the side set.

The following code segment will write a side set to an open EXODUS II file:

```
int error, exoid, id, num_sides, num_df, elem_list[2], side_list[2];  
float dist_fact[4];  
  
/* write side set parameters */  
  
id = 30;  
num_sides = 2;  
num_df = 4;  
  
error = ex_put_side_set_param (exoid, id, num_sides, num_df);  
  
/* write side set element and side lists */  
elem_list[0] = 1; elem_list[1] = 2;
```

```

side_list[0] = 1; side_list[1] = 1;

error = ex_put_side_set (exoid, id, elem_list, side_list);

/* write side set distribution factors */

dist_fact[0] = 30.0; dist_fact[1] = 30.1;
dist_fact[2] = 30.2; dist_fact[3] = 30.3;

error = ex_put_side_set_dist_fact (exoid, id, dist_fact);

```

## EXPSP: Fortran Interface

```
SUBROUTINE EXPSP (IDEXO, IDESS, NSESS, NDESS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER IDESS (R)
```

The side set ID.

```
INTEGER NSESS (R)
```

The number of sides (faces or edges) in the side set.

```
INTEGER NDESS (R)
```

The number of distribution factors on the side set.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will write a side set to an open EXODUS II file:

```

integer elem_list(2), side_list(2)
real dist_fact(4)

id = 31
numsid = 2
numdf = 4
elem_list(1) = 13
elem_list(2) = 14

side_list(1) = 3
side_list(2) = 4

dist_fact(1) = 31.0
dist_fact(2) = 31.1
dist_fact(3) = 31.2
dist_fact(4) = 31.3

call expsp (idexo, id, numsid, numdf, ierr)
call expss (idexo, id, elem_list, side_list, ierr)
call expssd (idexo, id, dist_fact, ierr)

```

## 4.2.28 Read Side Set Parameters

The function `ex_get_side_set_param` (or `EXGSP` for Fortran) reads the number of sides (faces on 3-d element types; edges on 2-d element types) which describe a single side set, and the number of distribution factors on the side set.

In case of an error, `ex_get_side_set_param` returns a negative number; a warning will return a positive number. `EXGSP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no side sets are stored in the file.
- incorrect side set ID.

### `ex_get_side_set_param`: C Interface

```
int ex_get_side_set_param (exoid, side_set_id,  
                          num_side_in_set, num_dist_fact_in_set);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int side\_set\_id (R)

The side set ID.

int\* num\_side\_in\_set (W)

Returned number of sides (faces or edges) in the side set.

int\* num\_dist\_fact\_in\_set (W)

Returned number of distribution factors on the side set.

The following coding will read all of the side sets from an open EXODUS II file:

```
int num_side_sets, error, exoid, num_sides_in_set, num_df_in_set,  
    num_elem_in_set, *ids, *elem_list, *side_list, *ctr_list, *node_list;  
float *dist_fact;  
  
error = ex_inq (exoid, EX_INQ_SIDE_SETS, &num_side_sets, &fdum, cdum);  
  
ids = (int *) calloc(num_side_sets, sizeof(int));  
error = ex_get_side_set_ids (exoid, ids);  
  
for (i=0; i<num_side_sets; i++)  
{  
    error = ex_get_side_set_param (exoid, ids[i], &num_sides_in_set,  
                                  &num_df_in_set);  
  
    num_elem_in_set = num_sides_in_set;  
    elem_list = (int *) calloc(num_elem_in_set, sizeof(int));  
    side_list = (int *) calloc(num_sides_in_set, sizeof(int));  
    error = ex_get_side_set (exoid, ids[i], elem_list, side_list);  
}
```

```

        if (num_df_in_set > 0)
        {
/* get side set node list to correlate to dist factors */
        ctr_list = (int *) calloc(num_elem_in_set, sizeof(int));
        node_list = (int *) calloc(num_df_in_set, sizeof(int));
        dist_fact = (float *) calloc(num_df_in_set, sizeof(float));

        error = ex_get_side_set_node_list (exoid, ids[i], ctr_list,
            node_list);
        error = ex_get_side_set_dist_fact (exoid, ids[i], dist_fact);
    }
}

```

## EXGSP: Fortran Interface

```

SUBROUTINE EXGSP (IDEXO, IDESS, NSESS, NDESS, IERR)

```

```

INTEGER IDEXO (R)

```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```

INTEGER IDESS (R)

```

The side set ID.

```

INTEGER NSESS (W)

```

Returned number of sides (faces or edges) in the side set.

```

INTEGER NDESS (W)

```

Returned number of distribution factors on the side set.

```

INTEGER IERR (W)

```

Returned error code. If no errors occurred, 0 is returned.

The following coding will read all of the side sets from an open EXODUS II file:

```

c NOTE:      MAXSS is the maximum number of side sets
c            MAXSID is the maximum number of sides in a side set
c            MAXNOD is the maximum number of nodes on a side set

        integer ids(MAXSS), numsid, numdf, elem1st(MAXSID), sidelst(MAXSID),
1         incnt(MAXSID), nodelst(MAXNOD)
        real distfact(MAXNOD)

        if (num_side_sets .gt. 0) then
            call exgssi (idexo, ids, ierr)
        endif

        do 10 i = 1, num_side_sets
            call exgsp (idexo, ids(i), numsid, numdf, ierr)
            call exgss (idexo, ids(i), elem1st, sidelst, ierr)
            call exgssn (idexo, ids(i), incnt, nodelst, ierr)
            call exgssd (idexo, ids(i), distfact, ierr)
10     continue

```

## 4.2.29 Write Side Set

The function `ex_put_side_set` (or `EXPSS` for Fortran) writes the side set element list and side set side (face on 3-d element types; edge on 2-d element types) list for a single side set. The routine `ex_put_side_set_param` (`EXPSP` for Fortran) must be called before this function is invoked.

In case of an error, `ex_put_side_set` returns a negative number; a warning will return a positive number. `EXPSS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_side_set_param` (or `EXPSP` for Fortran) not called previously.

### `ex_put_side_set`: C Interface

```
int ex_put_side_set (exoid, side_set_id, side_set_elem_list,  
                    side_set_side_list);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int side\_set\_id (R)

The side set ID.

int\* side\_set\_elem\_list (R)

Array containing the elements in the side set. Internal element IDs are used in this list (see Section 3.5 on page 7).

int\* side\_set\_side\_list (R)

Array containing the sides (faces or edges) in the side set.

For an example of a code segment to write a side set, refer to the description for `ex_put_side_set_param`.

### EXPSS: Fortran Interface

```
SUBROUTINE EXPSS (IDEXO, IDESS, LTEESS, LTSESS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER IDESS (R)

The side set ID.

INTEGER LTEESS(\*) (R)

Array containing the elements in the side set. Internal element IDs are used in this list (see Section 3.5 on page 7).



INTEGER LTSESS(\*) (R)

Array containing the sides (faces or edges) in the side set.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of a code segment to write a side set, refer to the description for EXPSP.

### 4.2.30 Read Side Set

The function `ex_get_side_set` (or `EXGSS` for Fortran) reads the side set element list and side set side (face for 3-d element types; edge for 2-d element types) list for a single side set. Memory must be allocated for the element list and side list (both are `num_side_in_set` in length) before this function is invoked.

In case of an error, `ex_get_side_set` returns a negative number; a warning will return a positive number. `EXGSS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no side sets are stored in the file.
- incorrect side set ID.

#### `ex_get_side_set`: C Interface

```
int ex_get_side_set (exoid, side_set_id, side_set_elem_list,  
                    side_set_side_list);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int side_set_id (R)`

The side set ID.

`int* side_set_elem_list (W)`

Returned array containing the elements in the side set. Internal element IDs are used in this list (see Section 3.5 on page 7).

`int* side_set_side_list (W)`

Returned array containing the sides (faces or edges) in the side set.

For an example of code to read a side set from an EXODUS II file, refer to the description for `ex_get_side_set_param`.

#### EXGSS: Fortran Interface

```
SUBROUTINE EXGSS (IDEXO, IDESS, LTEESS, LTSESS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER IDESS (R)

The side set ID.

INTEGER LTEESS(\*) (W)

Returned array containing the elements in the side set. Internal element IDs are used in this list (see Section 3.5 on page 7).

INTEGER LTSESS(\*) (W)

Returned array containing the faces (or edges) in the side set.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of code to read a side set from an EXODUS II file, refer to the description for EXGSP.

## 4.2.31 Write Side Set Distribution Factors

The function `ex_put_side_set_dist_fact` (or `EXPSSD` for Fortran) writes distribution factors for a single side set. The routine `ex_put_side_set_param` (or `EXPSP` for Fortran) must be called before this function is invoked.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_side_set_dist_fact` returns a negative number; a warning will return a positive number. `EXPSSD` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_side_set_param` (or `EXPSP` for Fortran) not called previously.
- a call to `ex_put_side_set_param` (or `EXPSP` for Fortran) specified zero distribution factors.

### `ex_put_side_set_dist_fact`: C Interface

```
int ex_put_side_set_dist_fact (exoid, side_set_id,  
                             side_set_dist_fact);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int side\_set\_id (R)

The side set ID.

void\* side\_set\_dist\_fact (R)

Array containing the distribution factors in the side set.

For an example of a code segment to write side set distribution factors, refer to the description for `ex_put_side_set_param`.

### **EXPSSD: Fortran Interface**

```
SUBROUTINE EXPSSD (IDEXO, IDESS, FACCESS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER IDESS (R)

The side set ID.

REAL FACESS( \* ) (R)

Array containing the distribution factors in the side set.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of a code segment to write side set distribution factors, refer to the description for EXPSP.

## 4.2.32 Read Side Set Distribution Factors

The function `ex_get_side_set_dist_fact` (or `EXGSSD` for Fortran) returns the distribution factors for a single side set. Memory must be allocated for the list of distribution factors (`num_dist_fact_in_set` in length) before this function is invoked.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_side_set_dist_fact` returns a negative number; a warning will return a positive number. `EXGSSD` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- a warning value is returned if no distribution factors were stored.

### `ex_get_side_set_dist_fact`: C Interface

```
int ex_get_side_set_dist_fact (exoid, side_set_id,  
    side_set_dist_fact);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int side\_set\_id (R)

The side set ID.

void\* side\_set\_dist\_fact (W)

Returned array containing the distribution factors in the side set.

For an example of code to read side set distribution factors from an EXODUS II file, refer to the description for `ex_get_side_set_param`.

### `EXGSSD`: Fortran Interface

```
SUBROUTINE EXGSSD (IDEXO, IDESS, FACESS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER IDESS (R)

The side set ID.

REAL FACESS(\*) (W)

Returned array containing the distribution factors in the side set.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of code to read side set distribution factors from an EXODUS II file, refer to the description for `EXGSP`.

### 4.2.33 Read Side Sets IDs

The function `ex_get_side_set_ids` (or `EXGSSI` for Fortran) reads the IDs of all of the side sets. Memory must be allocated for the returned array of (`num_side_sets`) IDs before this function is invoked.

In case of an error, `ex_get_side_set_ids` returns a negative number; a warning will return a positive number. `EXGSSI` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no side sets are stored in the file.

#### **ex\_get\_side\_set\_ids: C Interface**

```
int ex_get_side_set_ids (exoid, side_set_ids);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* side_set_ids (W)`

Returned array of the side sets IDs. The order of the IDs in this array reflects the sequence the side sets were introduced into the file.

For an example of code to read side set IDs from an EXODUS II file, refer to the description for `ex_get_side_set_param`.

#### **EXGSSI: Fortran Interface**

```
SUBROUTINE EXGSSI (IDEXO, IDESSS, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER IDESSS(*) (W)`

Returned array of side sets IDs. The order of the IDs in this array reflects the sequence the side sets were introduced into the file.

`INTEGER IERR (W)`

Returned error code. If no errors occurred, 0 is returned.

For an example of code to read side set IDs from an EXODUS II file, refer to the description for `EXGSP`.

## 4.2.34 Read Side Set Node List

The function `ex_get_side_set_node_list` (or `EXGSSN` for Fortran) returns a node count array and a list of nodes on a single side set. With the 2.0 and later versions of the database, this node list isn't stored directly but can be derived from the element number in the side set element list, local side number in the side set side list, and the element connectivity array. The application program must allocate memory for the node count array and node list.

There is a one-to-one mapping (i.e., same order -- as shown in Table 2, "Side Set Node Ordering," on page 16 -- and same number) between the nodes in the side set node list and the side set distribution factors. Thus, if distribution factors are stored for the side set of interest, the required size for the node list is the number of distribution factors returned by `ex_get_side_set_param` (or `EXGSP` for Fortran). If distribution factors are not stored for the side set, the application program must allocate a maximum size anticipated for the node list. This would be the product of the number of elements in the side set and the maximum number of nodes per side for all types of elements in the model, since side sets can span across different element types.

The length of the node count array is the length of the side set element list. For each entry in the side set element list, there is an entry in the side set side list, designating a local side number. The corresponding entry in the node count array is the number of nodes which define the particular side. In conjunction with the side set node list, this node count array gives an unambiguous nodal description of the side set.

In case of an error, `ex_get_side_set_node_list` returns a negative number; a warning will return a positive number. `EXGSSN` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no side sets are stored in the file.
- incorrect side set ID.

### `ex_get_side_set_node_list`: C Interface

```
int ex_get_side_set_node_list (exoid, side_set_id,  
    side_set_node_cnt_list, side_set_node_list);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int side_set_id (R)`

The side set ID.

`int* side_set_node_cnt_list (W)`

Returned array containing the number of nodes for each side (face in 3-d, edge in 2-d) in the side set.



`int* side_set_node_list (W)`

Returned array containing a list of nodes on the side set. Internal node IDs are used in this list (see Section 3.5 on page 7).

For an example of code to read a side set node list from an EXODUS II file, refer to the description for `ex_get_side_set_param`.

## **EXGSSN: Fortran Interface**

```
SUBROUTINE EXGSSN (IDEXO, IDESS, INCNT, LTNESS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

```
INTEGER IDESS (R)
```

The side set ID.

```
INCNT(*) (W)
```

Returned array containing the number of nodes for each side (face in 3-d, edge in 2-d) in the side set.

```
INTEGER LTNESS(*) (W)
```

Returned array containing a list of nodes on the side set. Internal node IDs are used in this list (see Section 3.5 on page 7).

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

For an example of code to read a side set node list from an EXODUS II file, refer to the description for `EXGSP`.

## 4.2.35 Write Concatenated Side Sets

The function `ex_put_concat_side_sets` (or `EXPCSS` for Fortran) writes the side set IDs, side set element count array, side set distribution factor count array, side set element pointers array, side set distribution factors pointers array, side set element list, side set side list, and side set distribution factors. “Concatenated side sets” refers to the arrays needed to define all of the side sets (ID array, side counts array, node counts array, element pointer array, node pointer array, element list, node list, and distribution factors array) as described in Section 3.12 on page 15. Writing concatenated side sets is more efficient than writing individual side sets. See Appendix A for a discussion of efficiency issues.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_concat_side_sets` returns a negative number; a warning will return a positive number. `EXPCSS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- the number of side sets specified in a call to `ex_put_init` (`EXPINI` for Fortran) was zero or has been exceeded.
- a side set with the same ID has already been stored.

### `ex_put_concat_side_sets`: C Interface

```
int ex_put_concat_side_sets (exoid, side_sets_ids,
                             num_side_per_set, num_dist_per_set,
                             side_sets_elem_index, side_sets_dist_index,
                             side_sets_elem_list, side_sets_side_list,
                             side_sets_dist_fact);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* side_sets_ids (R)`

Array containing the side set ID for each set.

`int* num_side_per_set (R)`

Array containing the number of sides for each set.

`int* num_dist_per_set (R)`

Array containing the number of distribution factors for each set.

`int* side_sets_elem_index (R)`  
 Array containing the indices into the `side_sets_elem_list` which are the locations of the first element for each set. These indices are 0-based.

`int* side_sets_dist_index (R)`  
 Array containing the indices into the `side_sets_dist_fact` which are the locations of the first distribution factor for each set. These indices are 0-based.

`int* side_sets_elem_list (R)`  
 Array containing the elements for all side sets. Internal element IDs are used in this list (see Section 3.5 on page 7).

`int* side_sets_side_list (R)`  
 Array containing the sides for all side sets.

`void* side_sets_dist_fact (R)`  
 Array containing the distribution factors for all side sets.

The following coding will write out two side sets in a concatenated format:

```
int error, exoid, ids[2], num_side_per_set[2], elem_ind[2],
    num_df_per_set[2], df_ind[2], elem_list[4], side_list[4];
float dist_fact[8];

/* write concatenated side sets */
ids[0] = 30;
ids[1] = 31;

num_side_per_set[0] = 2;
num_side_per_set[1] = 2;

elem_ind[0] = 0;
elem_ind[1] = 2;

num_df_per_set[0] = 4;
num_df_per_set[1] = 4;

df_ind[0] = 0;
df_ind[1] = 4;

/* side set #1 */
elem_list[0] = 2; elem_list[1] = 2;
side_list[0] = 2; side_list[1] = 1;
dist_fact[0] = 30.0; dist_fact[1] = 30.1;
dist_fact[2] = 30.2; dist_fact[3] = 30.3;

/* side set #2 */
elem_list[2] = 1; elem_list[3] = 2;
side_list[2] = 4; side_list[3] = 3;
dist_fact[4] = 31.0; dist_fact[5] = 31.1;
dist_fact[6] = 31.2; dist_fact[7] = 31.3;

error = ex_put_concat_side_sets (exoid, ids, num_side_per_set,
    num_df_per_set, elem_ind, df_ind, elem_list, side_list, dist_fact);
```

## EXPCSS: Fortran Interface

```
SUBROUTINE EXPCSS (IDEXO, IDESSS, NSESS, NDESS, IXEESS,  
                  IXDESS, LTEESS, LTSESS, FACESS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER IDESSS(*) (R)
```

Array containing the side set ID for each set.

```
INTEGER NSESS(*) (R)
```

Array containing the number of sides for each set.

```
INTEGER NDESS(*) (R)
```

Array containing the number of distribution factors for each set.

```
INTEGER IXEESS(*) (R)
```

Array containing the indices into the LTEESS array which are the locations of the first element for each set. These indices are 1-based.

```
INTEGER IXDESS(*) (R)
```

Array containing the indices into the FACESS array which are the locations of the first distribution factor for each set. These indices are 1-based.

```
INTEGER LTEESS(*) (R)
```

Array containing the elements for all side sets. Internal element IDs are used in this list (see Section 3.5 on page 7).

```
INTEGER LTSESS(*) (R)
```

Array containing the sides for all side sets.

```
REAL FACESS(*) (R)
```

Array containing the distribution factors for all side sets.

```
INTEGER IERR (R)
```

Returned error code. If no errors occurred, 0 is returned.

The following coding will write out two side sets in a concatenated format:

```
      integer ids(2), num_side_per_set(2), num_df_per_set(2),  
1      elem_ind(2), df_ind(2), elem_list(4), side_list(4)  
      real dist_fact(8)  
c  
c write concatenated side sets  
c  
  
      ids(1) = 30  
      ids(2) = 31  
  
      num_side_per_set(1) = 2  
      num_side_per_set(2) = 2  
  
      num_df_per_set(1) = 4
```

```

num_df_per_set(2) = 4

elem_ind(1) = 1
elem_ind(2) = 3

df_ind(1) = 1
df_ind(2) = 5
c
c side set #1 (ID of 30)
c
elem_list(1) = 11
elem_list(2) = 12

side_list(1) = 1
side_list(2) = 2

dist_fact(1) = 30.0
dist_fact(2) = 30.1
dist_fact(3) = 30.2
dist_fact(4) = 30.3
c
c side set #2 (ID of 31)
c
elem_list(3) = 13
elem_list(4) = 14

side_list(3) = 3
side_list(4) = 4

dist_fact(5) = 31.0
dist_fact(6) = 31.1
dist_fact(7) = 31.2
dist_fact(8) = 31.3

call expcss (idexo, ids, num_side_per_set, num_df_per_set,
1 elem_ind, df_ind, elem_list, side_list, dist_fact, ierr)

```

## 4.2.36 Read Concatenated Side Sets

The function `ex_get_concat_side_sets` (or `EXGCSS` for Fortran) reads the side set IDs, side set element count array, side set distribution factors count array, side set element pointers array, side set distribution factors pointers array, side set element list, side set side list, and side set distribution factors. “Concatenated side sets” refers to the arrays needed to define all of the side sets (ID array, side counts array, node counts array, element pointer array, node pointer array, element list, node list, and distribution factors array) as described in Section 3.12 on page 15.

Because the distribution factors are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

The length of each of the returned arrays can be determined by invoking `ex_inquire` (or `EXINQ` for Fortran). See Section 4.1.11 on page 41.

In case of an error, `ex_get_concat_side_sets` returns a negative number; a warning will return a positive number. `EXGCSS` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- a warning value is returned if no side sets are stored in the file.

### `ex_get_concat_side_sets`: C Interface

```
int ex_get_concat_side_sets (exoid, side_set_ids,
    num_side_per_set, num_dist_per_set,
    side_sets_elem_index, side_sets_dist_index,
    side_sets_elem_list, side_sets_side_list,
    side_sets_dist_fact);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* side_set_ids (W)`

Returned array containing the side set ID for each set.

`int* num_side_per_set (W)`

Returned array containing the number of sides for each set.

`int* num_dist_per_set (W)`

Returned array containing the number of distribution factors for each set.

`int* side_sets_elem_index (W)`

Returned array containing the indices into the `side_sets_elem_list` which are the locations of the first element for each set. These indices are 0-based.

`int* side_sets_dist_index (W)`

Returned array containing the indices into the `side_sets_dist_fact` array which are the locations of the first distribution factor for each set. These indices are 0-based.

`int* side_sets_elem_list (W)`

Returned array containing the elements for all side sets. Internal element IDs are used in this list (see Section 3.5 on page 7).

`int* side_sets_side_list (W)`

Returned array containing the sides for all side sets.

`void* side_sets_dist_fact (W)`

Returned array containing the distribution factors for all side sets.

The following code segment will return in concatenated format all the side sets stored in an EXODUS II file:

```
#include "exodusII.h"
int error, exoid, num_ss, elem_list_len, df_list_len, *ids, *side_list,
    *num_side_per_set, *num_df_per_set, *elem_ind, *df_ind, *elem_list;
float *dist_fact;

error = ex_inquire (exoid, EX_INQ_SIDE_SETS, &num_ss, &fdum, cdum);
if (num_ss > 0) {
    error = ex_inquire(exoid, EX_INQ_SS_ELEM_LEN, &elem_list_len, &fdum,
        cdum);

    error = ex_inquire(exoid, EX_INQ_SS_DF_LEN, &df_list_len, &fdum,
        cdum);

    /* read concatenated side sets */

    ids = (int *) calloc(num_ss, sizeof(int));
    num_side_per_set = (int *) calloc(num_ss, sizeof(int));
    num_df_per_set = (int *) calloc(num_ss, sizeof(int));
    elem_ind = (int *) calloc(num_ss, sizeof(int));
    df_ind = (int *) calloc(num_ss, sizeof(int));
    elem_list = (int *) calloc(elem_list_len, sizeof(int));
    side_list = (int *) calloc(elem_list_len, sizeof(int));
    dist_fact = (float *) calloc(df_list_len, sizeof(float));

    error = ex_get_concat_side_sets (exoid, ids, num_side_per_set,
        num_df_per_set, elem_ind, df_ind, elem_list, side_list, dist_fact);
}
```

## EXGCSS: Fortran Interface

```
SUBROUTINE EXGCSS (IDEXO, IDESSS, NSESS, NDESS, IXEESS,
    IXDESS, LTEESS, LTSESS, FACESS, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IDESSS(\*) (W)  
Returned array containing the side set ID for each set.

INTEGER NSESS(\*) (W)  
Returned array containing the number of sides for each set.

INTEGER NDESS(\*) (W)  
Returned array containing the number of distribution factors for each set.

INTEGER IXEESS(\*) (W)  
Returned array containing the indices into the LTEESS array which are the locations of the first element for each set. These indices are 1-based.

INTEGER IXDESS(\*) (W)  
Returned array containing the indices into the FACESS array which are the locations of the first distribution factor for each set. These indices are 1-based.

INTEGER LTEESS(\*) (W)  
Returned array containing the elements for all side sets. Internal element IDs are used in this list (see Section 3.5 on page 7).

INTEGER LTSESS(\*) (W)  
Returned array containing the sides for all side sets.

REAL FACESS(\*) (W)  
Returned array containing the distribution factors for all side sets.

INTEGER IERR (W)  
Returned error code. If no errors occurred, 0 is returned.

The following code segment will return in concatenated format all the side sets stored in an EXODUS II file:

```

c NOTE:      MAXSS is the maximum number of side sets
c           MAXSID is the maximum number of sides in a side set
c           MAXDF is the max number of distribution factors in a side set

      integer elemlen, nodelen, dflen, ids(MAXSS), num_side(MAXSS),
1         num_df(MAXSS), elem_ind(MAXSS), df_ind(MAXSS),
2         elem_list(MAXSID*MAXSS), side_list(MAXSID*MAXSS)
      real dist_fact(MAXDF*MAXSS)

      call exinq (idexo, EXSIDS, num_side_sets, fdum, cdum, ierr)

      if (num_side_sets .gt. 0) then
c
c use the following inquiries if dynamic allocation is available
      call exinq (idexo, EXSSEL, elemlen, fdum, cdum, ierr)
      call exinq (idexo, EXSSNL, nodelen, fdum, cdum, ierr)
      call exinq (idexo, EXSSDF, dflen, fdum, cdum, ierr)
c
c read concatenated side sets
      call exgcss (idexo, ids, num_side, num_df, elem_ind, df_ind,
        elem_list, side_list, dist_fact, ierr)
      endif

```



## 4.2.37 Convert Side Set Nodes to Sides

The function `ex_cvt_nodes_to_sides` (or `EXCN2S` for Fortran) is used to convert a side set node list to a side set side list. This routine is provided for application programs that utilize side sets defined by nodes (as was done previous to release 2.0) rather than local faces or edges. The application program must allocate memory for the returned array of sides. The length of this array is the same as the length of the concatenated side sets element list, which can be determined with a call to `ex_inquire` (or `EXINQ` for Fortran).

In case of an error, `ex_cvt_nodes_to_sides` returns a negative number; a warning will return a positive number. `EXCN2S` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- a warning value is returned if no side sets are stored in the file.
- because the faces of a wedge require a different number of nodes to describe them (quadrilateral vs. triangular faces), the function will abort with a fatal return code if a wedge is encountered in the side set element list.

### `ex_cvt_nodes_to_sides`: C Interface

```
int ex_cvt_nodes_to_sides (exoid, num_side_per_set,
                          num_nodes_per_set, side_sets_elem_index,
                          side_sets_node_index, side_sets_elem_list,
                          side_sets_node_list, side_sets_side_list);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int* num_side_per_set (R)`

Array containing the number of sides for each set. The number of sides is equal to the number of elements for each set.

`int* num_nodes_per_set (R)`

Array containing the number of nodes for each set.

`int* side_sets_elem_index (R)`

Array containing indices into the `side_sets_elem_list` which are the locations of the first element for each set. These indices are 0-based.

`int* side_sets_node_index (R)`

Array containing indices into the `side_sets_node_list` which are the locations of the first node for each set. These indices are 0-based.

`int* side_sets_elem_list (R)`

Array containing the elements for all side sets. Internal element IDs are used in this list (see Section 3.5 on page 7).

`int* side_sets_node_list (R)`

Array containing the nodes for all side sets. Internal node IDs are used in this list (see Section 3.5 on page 7).

```
int* side_sets_side_list (W)
```

Returned array containing the sides for all side sets.

The following code segment will convert side sets described by nodes to side sets described by local side numbers:

```
int error, exoid, ids[2], num_side_per_set[2], num_nodes_per_set[2],
    elem_ind[2], node_ind[2], elem_list[4], node_list[8], el_lst_len,
    *side_list;

ids[0] = 30; ids[1] = 31;

num_side_per_set[0] = 2; num_side_per_set[1] = 2;
num_nodes_per_set[0] = 4; num_nodes_per_set[1] = 4;
elem_ind[0] = 0; elem_ind[1] = 2;
node_ind[0] = 0; node_ind[1] = 4;

/* side set #1 */
elem_list[0] = 2; elem_list[1] = 2;
node_list[0] = 8; node_list[1] = 5; node_list[2] = 6; node_list[3] = 7;

/* side set #2 */
elem_list[2] = 1; elem_list[3] = 2;
node_list[4] = 2; node_list[5] = 3; node_list[6] = 7; node_list[7] = 8;

error = ex_inquire (exoid, EX_INQ_SS_ELEM_LEN, &el_lst_len, &fdum,cdum);

/* side set element list is same length as side list */
side_list = (int *) calloc (el_lst_len, sizeof(int));

ex_cvt_nodes_to_sides(exoid, num_side_per_set, num_nodes_per_set,
    elem_ind, node_ind, elem_list, node_list, side_list);
```

## EXCN2S: Fortran Interface

```
SUBROUTINE EXCN2S(IDEXO, NSESS, NDESS, IXEESS, IXNESS,
    LTEESS, LTNESS, LTSESS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER NSESS(*) (R)
```

Array containing the number of sides for each set. The number of sides is equal to the number of elements for each set.

```
INTEGER NDESS(*) (R)
```

Array containing the number of nodes for each set.

```
INTEGER IXEESS(*) (R)
```

Array containing indices into the LTEESS array which are the locations of the first element for each set. These indices are 1-based.

INTEGER IXNESS(\*) (R)

Array containing indices into the LTNESS array which are the locations of the first node for each set. These indices are 1-based.

INTEGER LTEESS(\*) (R)

Array containing the elements for all side sets. Internal element IDs are used in this list (see Section 3.5 on page 7).

INTEGER LTNESS(\*) (R)

Array containing the nodes for all side sets. Internal node IDs are used in this list (see Section 3.5 on page 7).

INTEGER LTSESS(\*) (W)

Returned array containing the sides for all side sets.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code segment will convert side sets described by nodes to side sets described by local side numbers:

```
    INCLUDE 'exodusII.inc'
    integer ids(2), num_side_per_set(2), num_nodes_per_set(2),
1      elem_ind(2), node_ind(2), node_list(8), elem_list(4),
2      side_list(4)

    ids(1) = 30
    ids(2) = 31
    num_side_per_set(1) = 2
    num_side_per_set(2) = 2
    num_nodes_per_set(1) = 4
    num_nodes_per_set(2) = 4
    elem_ind(1) = 1
    elem_ind(2) = 3
    node_ind(1) = 1
    node_ind(2) = 5
c
c side set #1
    node_list(1) = 8
    node_list(2) = 5
    node_list(3) = 6
    node_list(4) = 7
    elem_list(1) = 2
    elem_list(2) = 2
c
c side set #2
    node_list(5) = 2
    node_list(6) = 3
    node_list(7) = 7
    node_list(8) = 8
    elem_list(3) = 1
    elem_list(4) = 2
    call excn2s(idexo, num_side_per_set, num_nodes_per_set, elem_ind,
1      node_ind, elem_list, node_list, side_list, ierr)
```

## 4.2.38 Write Property Arrays Names

The function `ex_put_prop_names` (or `EXPPN` for Fortran) writes property names and allocates space for property arrays used to assign integer properties to element blocks, node sets, or side sets. The property arrays are initialized to zero (0). Although this function is optional, since `ex_put_prop` will allocate space within the data file if it hasn't been previously allocated, it is more efficient to use `ex_put_prop_names` if there is more than one property to store. See Appendix A for a discussion of efficiency issues.

In case of an error, `ex_put_prop_names` returns a negative number; a warning will return a positive number. `EXPPN` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- invalid object type specified.
- no object of the specified type is stored in the file.

### `ex_put_prop_names`: C Interface

```
int ex_put_prop_names (exoid, obj_type, num_props,  
                      prop_names);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int obj_typ (R)`

Type of object; use one of the following options:

- `EX_ELEM_BLOCK`            To designate an element block.
- `EX_NODE_SET`             To designate a node set.
- `EX_SIDE_SET`             To designate a side set.

`int num_props (R)`

The number of integer properties to be assigned to all of the objects of the type specified (element blocks, node sets, or side sets).

`char** prop_names (R)`

Array containing `num_props` names (of maximum length of `MAX_STR_LENGTH`) of properties to be stored.

For instance, suppose a user wanted to assign the 1st, 3rd, and 5th element blocks (those element blocks stored 1st, 3rd, and 5th, regardless of their ID) to a group (property) called "TOP", and the 2nd, 3rd, and 4th element blocks to a group called "LSIDE". This could be accomplished with the following code:

```
#include "exodusII.h";  
char* prop_names[2];  
int top_part[] = {1,0,1,0,1};  
int lside_part[] = {0,1,1,1,0};
```

```

int id[] = {10, 20, 30, 40, 50};
prop_names[0] = "TOP";
prop_names[1] = "LSIDE";
/* This call to ex_put_prop_names is optional, but more efficient */
ex_put_prop_names (exoid, EX_ELEM_BLOCK, 2, prop_names);

/* The property values can be output individually thus */
for (i=0; i<5; i++){
    ex_put_prop (exoid, EX_ELEM_BLOCK, id[i], prop_names[0],
                top_part[i]);
    ex_put_prop (exoid, EX_ELEM_BLOCK, id[i], prop_names[1],
                lside_part[i]); }

/* Alternatively, the values can be output as an array thus*/
ex_put_prop_array (exoid, EX_ELEM_BLOCK, prop_names[0], top_part);
ex_put_prop_array (exoid, EX_ELEM_BLOCK, prop_names[1], lside_part);

```

## EXPPN: Fortran Interface

```
SUBROUTINE EXPPN (IDEXO, ITYPE, NPROPS, NAMEPR, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER ITYPE (R)

Type of object; use one of the following options:

- EXEBLK           To designate an element block.
- EXNSET           To designate a node set.
- EXSSET           To designate a side set.

INTEGER NPROPS (R)

The number of integer properties to be assigned to all of the objects of the type specified (element blocks, node sets, or side sets).

CHARACTER\*MXSTLN NAMEPR(\*) (R)

Array containing NPROPS names of properties to be stored.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following example assigns a property "STEEL" to the first and third element blocks with ID's 10 and 30, respectively.

```

include 'exodusII.inc'
integer ival(3)
data ival/1,0,1/
C This call to EXPPN is optional, but more efficient
call exppn (idexo, exeblk, 1, "STEEL", ierr)

C The property values can be written individually thus
call expp (idexo, EXEBLK, 10, "STEEL", 1, ierr)
call expp (idexo, EXEBLK, 30, "STEEL", 1, ierr)
c Alternatively, the values can be written as an array thus
call exppa (idexo, EXEBLK, "STEEL", ival, ierr)

```

## 4.2.39 Read Property Arrays Names

The function `ex_get_prop_names` (or `EXGPN` for Fortran) returns names of integer properties stored for an element block, node set, or side set. The number of properties (needed to allocate space for the property names) can be obtained via a call to `ex_inquire` (`EXINQ` for Fortran). See Section 4.1.11 on page 41.

In case of an error, `ex_get_prop_names` returns a negative number; a warning will return a positive number. `EXGPN` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- invalid object type specified.

### `ex_get_prop_names`: C Interface

```
int ex_get_prop_names (exoid, obj_type, prop_names);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int obj\_type (R)

Type of object; use one of the following options:

- `EX_ELEM_BLOCK`            To designate an element block.
- `EX_NODE_SET`             To designate a node set.
- `EX_SIDE_SET`             To designate a side set.

char\*\* prop\_names (W)

Returned array containing `num_props` (obtained from call to `ex_inquire`) names (of maximum length `MAX_STR_LENGTH`) of properties to be stored. "ID", a reserved property name, will be the first name in the array.

As an example, the following code segment reads in properties assigned to node sets:

```
#include "exodusII.h";
int error, exoid, num_props, *prop_values;
char *prop_names[MAX_PROPS];

/* read node set properties */
error = ex_inquire (exoid, EX_INQ_NS_PROP, &num_props, &fdum, cdum);

for (i=0; i<num_props; i++){
    prop_names[i] = (char *) malloc ((MAX_STR_LENGTH+1), sizeof(char));
    prop_values = (int *) malloc (num_node_sets, sizeof(int));

    error = ex_get_prop_names(exoid, EX_NODE_SET, prop_names);
    for (i=0; i<num_props; i++){
        error = ex_get_prop_array(exoid, EX_NODE_SET, prop_names[i],
            prop_values);
    }
}
```

## EXGPN: Fortran Interface

```
SUBROUTINE EXGPN (IDEXO, ITYPE, NAMEPR, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER ITYPE (R)
```

Type of object; use one of the following options:

- EXEBLK           To designate an element block.
- EXNSET           To designate a node set.
- EXSSET           To designate a side set.

```
CHARACTER*MXSTLN NAMEPR(*) (W)
```

Returned array containing NPROPS (obtained from call to EXINQ) names of properties to be stored. "ID", a reserved property name, will be the first name in the array.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

As an example, the following will read the side set property values from an EXODUS II file:

```
c NOTE:           MAXSS is the maximum number of side sets
c           MXSSPR is the maximum number of side set properties

      include 'exodusII.inc'
      integer ids(MAXSS), ivalS(MAXSS, MXSSPR)
      character*(MXSTLN) prop_names(MXSSPR)

c determine number of side sets and side set properties
      call exinq (idexo, EXSIDS, num_side_sets, fdum, cdum, ierr)
      call exinq (idexo, EXNSSP, num_props, fdum, cdum, ierr)

c get the side set property names
      call exgpn(idexo, EXSSET, prop_names, ierr)

c get the side set ids
      call exgssi (idexo, ids, ierr)

c get the side set property values individually
      do 20 i = 1, num_props
          do 10 j = 1, num_side_sets
              call exgp(idexo, EXSSET,ids(j),prop_names(i),ivalS(j,i),ierr)
10            continue
20            continue

c alternatively, the property values can be read in together as follows
      do 30 i = 1, num_props
          call exgpa (idexo, EXSSET, prop_names(i), ivalS(1,i), ierr)
30            continue
```

## 4.2.40 Write Object Property

The function `ex_put_prop` (or `EXPP` for Fortran) stores an integer property value to a single element block, node set, or side set. Although it is not necessary to invoke `ex_put_prop_names` (`EXPPN` for Fortran), since `ex_put_prop` will allocate space within the data file if it hasn't been previously allocated, it is more efficient to use `ex_put_prop_names` if there is more than one property to store. See Appendix A for a discussion of efficiency issues.

It should be noted that the interpretation of the values of the integers stored as properties is left to the application code. In general, a zero (0) means the object does not have the specified property (or is not in the specified group); a nonzero value means the object does have the specified property. When space is allocated for the properties using `ex_put_prop_names` or `ex_put_prop`, the properties are initialized to zero (0).

Because the ID of an element block, node set, or side set is just another property (named "ID"), this routine can be used to change the value of an ID. This feature must be used with caution, though, because changing the ID of an object to the ID of another object of the same type (element block, node set, or side set) would cause two objects to have the same ID, and thus only the first would be accessible. Therefore, `ex_put_prop` issues a warning if a user attempts to give two objects the same ID.

In case of an error, `ex_put_prop` returns a negative number; a warning will return a positive number. `EXPP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- invalid object type specified.
- a warning is issued if a user attempts to change the ID of an object to the ID of an existing object of the same type.

### `ex_put_prop`: C Interface

```
int ex_put_prop (exoid, obj_type, obj_id, prop_name, value);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int obj_type (R)
```

Type of object; use one of the following options:

- `EX_ELEM_BLOCK`            To designate an element block.
- `EX_NODE_SET`              To designate a node set.
- `EX_SIDE_SET`              To designate a side set.

```
int obj_id (R)
```

The element block, node set, or side set ID.



char\* prop\_name (R)

The name of the property for which the value will be stored. Maximum length of this string is MAX\_STR\_LENGTH.

int value (R)

The value of the property.

For an example of code to write out an object property, refer to the description for ex\_put\_prop\_names.

## **EXPP: Fortran Interface**

```
SUBROUTINE EXPP (IDEXO, ITYPE, ID, NAMEPR, IVAL, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER ITYPE (R)

Type of object; use one of the following options:

- EXEBLK            To designate an element block.
- EXNSET            To designate a node set.
- EXSSET            To designate a side set.

INTEGER ID (R)

The element block, node set, or side set ID.

CHARACTER\*MXSTLN NAMEPR (R)

The name of the property for which a value will be stored.

INTEGER IVAL (R)

The value of the property.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of code to write out an object property, refer to the description for EXPPN.

## 4.2.41 Read Object Property

The function `ex_get_prop` (or `EXGP` for Fortran) reads an integer property value stored for a single element block, node set, or side set.

In case of an error, `ex_get_prop` returns a negative number; a warning will return a positive number. `EXGP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- invalid object type specified.
- a warning value is returned if a property with the specified name is not found.

### `ex_get_prop`: C Interface

```
int ex_get_prop (exoid, obj_type, obj_id, prop_name, value);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int obj_type (R)`

Type of object; use one of the following options:

- `EX_ELEM_BLOCK`            To designate an element block.
- `EX_NODE_SET`              To designate a node set.
- `EX_SIDE_SET`              To designate a side set.

`int obj_id (R)`

The element block, node set, or side set ID.

`char* prop_name (R)`

The name of the property (maximum length is `MAX_STR_LENGTH`) for which the value is desired.

`int* value (W)`

Returned value of the property.

For an example of code to read an object property, refer to the description for `ex_get_prop_names`.

### `EXGP`: Fortran Interface

```
SUBROUTINE EXGP (IDEXO, ITYPE, ID, NAMEPR, IVAL, IERR)
```

`INTEGER IDEXO (R)`

EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER ITYPE (R)

Type of object; use one of the following options:

- EXEBLK           To designate an element block.
- EXNSET           To designate a node set.
- EXSSET           To designate a side set.

INTEGER ID (R)

The element block, node set, or side set ID.

CHARACTER\*MXSTLN NAMEPR (R)

The name of the property for which the value is desired.

INTEGER IVAL (W)

Returned value of the property.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of code to read an object property, refer to the description for EXGPN.

## 4.2.42 Write Object Property Array

The function `ex_put_prop_array` (or `EXPPA` for Fortran) stores an array of (`num_elem_blk`, `num_node_sets`, or `num_side_sets`) integer property values for all element blocks, node sets, or side sets. The order of the values in the array must correspond to the order in which the element blocks, node sets, or side sets were introduced into the file. For instance, if the parameters for element block with ID 20 were written to a file (via `ex_put_elem_block`; or `EXPELB` for Fortran), and then parameters for element block with ID 10, followed by the parameters for element block with ID 30, the first, second, and third elements in the property array would correspond to element block 20, element block 10, and element block 30, respectively.

One should note that this same functionality (writing properties to multiple objects) can be accomplished with multiple calls to `ex_put_prop` (or `EXPP` in Fortran).

Although it is not necessary to invoke `ex_put_prop_names` (`EXPPN` for Fortran), since `ex_put_prop_array` will allocate space within the data file if it hasn't been previously allocated, it is more efficient to use `ex_put_prop_names` if there is more than one property to store. See Appendix A for a discussion of efficiency issues.

In case of an error, `ex_put_prop_array` returns a negative number; a warning will return a positive number. `EXPPA` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- invalid object type specified.

### `ex_put_prop_array`: C Interface

```
int ex_put_prop_array (exoid, obj_type, prop_name, values);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int obj_type (R)`

Type of object; use one of the following options:

- `EX_ELEM_BLOCK`            To designate an element block.
- `EX_NODE_SET`              To designate a node set.
- `EX_SIDE_SET`              To designate a side set.

`char* prop_name (R)`

The name of the property for which the values will be stored. Maximum length of this string is `MAX_STR_LENGTH`.

`int* values (R)`

An array of property values.

For an example of code to write an array of object properties, refer to the description for `ex_put_prop_names`.

## **EXPPA: Fortran Interface**

```
SUBROUTINE EXPPA (IDEXO, ITYPE, NAMEPR, IVALS, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER ITYPE (R)
```

Type of object; use one of the following options:

- EXEBLK           To designate an element block.
- EXNSET           To designate a node set.
- EXSSET           To designate a side set.

```
CHARACTER*MXSTLN NAMEPR (R)
```

The name of the property for which the values will be stored.

```
INTEGER IVAL(*) (R)
```

An array of property values.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

For an example of code to write an array of object properties, refer to the description for `EXPPN`.

## 4.2.43 Read Object Property Array

The function `ex_get_prop_array` (or `EXGPA` for Fortran) reads an array of integer property values for all element blocks, node sets, or side sets. The order of the values in the array correspond to the order in which the element blocks, node sets, or side sets were introduced into the file. Before this function is invoked, memory must be allocated for the returned array of (`num_elem_blk`, `num_node_sets`, or `num_side_sets`) integer values.

This function can be used in place of `ex_get_elem_blk_ids` (`EXGEBI` for Fortran), `ex_get_node_set_ids` (`EXGNSI` for Fortran), and `ex_get_side_set_ids` (`EXGSSI` for Fortran) to get element block, node set, and side set IDs, respectively, by requesting the property name “ID.” One should also note that this same function can be accomplished with multiple calls to `ex_get_prop` (or `EXGP` in Fortran).

In case of an error, `ex_get_prop_array` returns a negative number; a warning will return a positive number. `EXGPA` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- invalid object type specified.
- a warning value is returned if a property with the specified name is not found.

### `ex_get_prop_array`: C Interface

```
int ex_get_prop_array (exoid, obj_type, prop_name, values);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int obj_type
```

Type of object; use one of the following options:

- `EX_ELEM_BLOCK`            To designate an element block.
- `EX_NODE_SET`              To designate a node set.
- `EX_SIDE_SET`              To designate a side set.

```
char* prop_name (R)
```

The name of the property (maximum length of `MAX_STR_LENGTH`) for which the values are desired.

```
int* values (W)
```

Returned array of property values.

For an example of code to read an array of object properties, refer to the description for `ex_get_prop_names`.

## EXGPA: Fortran Interface

SUBROUTINE EXGPA (IDEXO, ITYPE, NAMEPR, IVALS, IERR)

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER ITYPE (R)

Type of object; use one of the following options:

- EXEBLK           To designate an element block.
- EXNSET           To designate a node set.
- EXSSET           To designate a side set.

CHARACTER\*MXSTLN NAMEPR (R)

The name of the property for which the values are desired.

INTEGER IVAL(\*) (W)

Returned array of property values.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For an example of code to read an array of object properties, refer to the description for EXGPN.

### **4.3 Results Data**

This section describes functions which read and write analysis results data and related entities. These include results variables (global, elemental, and nodal), element variable truth table, and simulation times.



### 4.3.1 Write Results Variables Parameters

The function `ex_put_var_param` (or `EXPVP` for Fortran) writes the number of global, nodal, or element variables that will be written to the database.

In case of an error, `ex_put_var_param` returns a negative number; a warning will return a positive number. `EXPVP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- invalid variable type specified (must be “g”, “G”, “n”, “N”, “e”, or “E”).
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- this routine has already been called with the same variable type; redefining the number of variables is not allowed.
- a warning value is returned if the number of variables is specified as zero.

#### `ex_put_var_param`: C Interface

```
int ex_put_var_param (exoid, var_type, num_vars);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
char* var_type (R)
```

Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

```
int num_vars (R)
```

The number of `var_type` variables that will be written to the database.

For example, the following code segment initializes the data file to store global variables:

```
int num_glo_vars, error, exoid;

/* write results variables parameters */

num_glo_vars = 3;
error = ex_put_var_param (exoid, "g", num_glo_vars);
```

## EXPVP: Fortran Interface

```
SUBROUTINE EXPVP (IDEXO, VARTYP, NVAR, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
CHARACTER*1 VARTYP (R)
```

Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

```
INTEGER NVAR (R)
```

The number of VARTYP variables that will be written to the database.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

For example, the following code segment initializes the data file to store global variables:

```
num_glo_vars = 1  
call expvp (idexo, "g", num_glo_vars, ierr)
```

## 4.3.2 Read Results Variables Parameters

The function `ex_get_var_param` (or `EXGVP` for Fortran) reads the number of global, nodal, or element variables stored in the database.

In case of an error, `ex_get_var_param` returns a negative number; a warning will return a positive number. `EXGVP` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- invalid variable type specified (must be “g”, “G”, “n”, “N”, “e”, or “E”).

### `ex_get_var_param`: C Interface

```
int ex_get_var_param (exoid, var_type, num_vars);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char* var_type (R)`  
Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

`int* num_vars (W)`  
Returned number of `var_type` variables that are stored in the database.

As an example, the following coding will determine the number of global variables stored in the data file:

```
int num_glo_vars, error, exoid;

/* read global variables parameters */

error = ex_get_var_param (exoid, "g", &num_glo_vars);
```

### `EXGVP`: Fortran Interface

```
SUBROUTINE EXGVP (IDEXO, VARTYP, NVAR, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

CHARACTER\*1 VARTYP (R)

Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

INTEGER NVAR (W)

Returned number of VARTYP variables that are stored in the database.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

As an example, the following coding will determine the number of global variables stored in the data file:

```
call exgvp (idexo, "g", num_glo_vars, ierr)
```

### 4.3.3 Write Results Variables Names

The function `ex_put_var_names` or (EXPVAN for Fortran) writes the names of the results variables to the database. The names are `MAX_STR_LENGTH`-characters in length. The function `ex_put_var_param` (EXPVP for Fortran) must be called before this function is invoked.

In case of an error, `ex_put_var_names` returns a negative number; a warning will return a positive number. EXPVAN returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file not initialized properly with call to `ex_put_init` (EXPINI for Fortran).
- invalid variable type specified (must be “g”, “G”, “n”, “N”, “e”, or “E”).
- `ex_put_var_param` (EXPVP for Fortran) was not called previously or was called with zero variables of the specified type.
- `ex_put_var_names` or (EXPVAN for Fortran) has been called previously for the specified variable type.

#### `ex_put_var_names`: C Interface

```
int ex_put_var_names (exoid, var_type, num_vars,  
                    var_names[]);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char* var_type (R)`

Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

`int num_vars (R)`

The number of `var_type` variables that will be written to the database.

`char** var_names (R)`

Array of pointers to `num_vars` variable names.

The following coding will write out the names associated with the nodal variables:

```
int num_nod_vars, error, exoid;  
char *var_names[2];  
  
/* write results variables parameters and names */  
  
num_nod_vars = 2;
```

```

var_names[0] = "disx";
var_names[1] = "disy";

error = ex_put_var_param (exoid, "n", num_nod_vars);
error = ex_put_var_names (exoid, "n", num_nod_vars, var_names);

```

## EXPVAN: Fortran Interface

```
SUBROUTINE EXPVAN (IDEXO, VARTYP, NVAR, NAMES, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
CHARACTER*1 VARTYP (R)
```

Character indicating the type of variable which is described. Use one of the following options:

- "g" (or "G") For global variables.
- "n" (or "N") For nodal variables.
- "e" (or "E") For element variables.

```
INTEGER NVAR (R)
```

The number of VARTYP variables that will be written to the database.

```
CHARACTER*MXSTLN NAMES (*)
```

Array containing NVAR variable names.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following coding will write out the names associated with the nodal variables:

```

include 'exodusII.inc'
character*(MXSTLN)var_names(1)

var_names(1) = "glo_vars"
call expvan (idexo, "g", num_glo_vars, var_names, ierr)

```

### 4.3.4 Read Results Variables Names

The function `ex_get_var_names` or (`EXGVAN` for Fortran) reads the names of the results variables from the database. Memory must be allocated for the name array before this function is invoked. The names are `MAX_STR_LENGTH`-characters in length.

In case of an error, `ex_get_var_names` returns a negative number; a warning will return a positive number. `EXGVAN` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- invalid variable type specified (must be “g”, “G”, “n”, “N”, “e”, or “E”).
- a warning value is returned if no variables of the specified type are stored in the file.

#### `ex_get_var_names`: C Interface

```
int ex_get_var_names (exoid, var_type, num_vars,
                    var_names[]);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`char* var_type`

Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

`int num_vars (R)`

The number of `var_type` variables that will be read from the database.

`char** var_names (W)`

Returned array of pointers to `num_vars` variable names.

As an example, the following code segment will read the names of the nodal variables stored in the data file:

```
#include "exodusII.h"
int error, exoid, num_nod_vars;
char *var_names[10];

/* read nodal variables parameters and names */

error = ex_get_var_param (exoid, "n", &num_nod_vars);

for (i=0; i<num_nod_vars; i++)
    var_names[i] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));
error = ex_get_var_names (exoid, "n", num_nod_vars, var_names);
```

## EXGVAN: Fortran Interface

```
SUBROUTINE EXGVAN (IDEXO, VARTYP, NVAR, NAMES, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
CHARACTER*1 VARTYP (R)
```

Character indicating the type of variable which is described. Use one of the following options:

- “g” (or “G”) For global variables.
- “n” (or “N”) For nodal variables.
- “e” (or “E”) For element variables.

```
INTEGER NVAR (R)
```

The number of VARTYP variables that will be read from the database.

```
CHARACTER*MXSTLN NAMES(*) (W)
```

Returned array containing NVAR (returned from a call to EXGVP) variable names.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

As an example, the following code segment will read the names of the global variables stored in the data file:

```
c NOTE:      MAXVARS is the maximum number of global variables
c
c      include 'exodusII.inc'
c      character*(MXSTLN) var_names(MAXVARS)
c
c read global variables parameters and names
c
c      call exgvp (idexo, "g", num_glo_vars, ierr)
c
c      call exgvn (idexo, "g", num_glo_vars, var_names, ierr)
```



### 4.3.5 Write Time Value for a Time Step

The function `ex_put_time` (or `EXPTIM` for Fortran) writes the time value for a specified time step.

Because time values are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_time` returns a negative number; a warning will return a positive number. `EXPTIM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.

#### **ex\_put\_time: C Interface**

```
int ex_put_time (exoid, time_step, time_value);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`  
The time step number. This is essentially a counter that is incremented only when results variables are output to the data file. The first time step is 1.

`void* time_value (R)`  
The time at the specified time step.

The following code segment will write out the simulation time value at simulation time step `n`:

```
int error, exoid, n;  
float time_value;  
  
/* write time value */  
  
error = ex_put_time (exoid, n, &time_value);
```

#### **EXPTIM: Fortran Interface**

```
SUBROUTINE EXPTIM (IDEXO, NSTEP, TIME, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

`INTEGER NSTEP (R)`  
The time step number. This essentially a counter that is incremented only when results variables are output to the data file. The first time step is 1.

REAL TIME (R)

The time at the specified time step.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following code segment will write out the simulation time value at simulation time step n:

```
c  
c write time value to file  
c  
  
    call exptim (idexo, n, time_value, ierr)
```

### 4.3.6 Read Time Value for a Time Step

The function `ex_get_time` (or `EXGTIM` for Fortran) reads the time value for a specified time step.

Because time values are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_time` returns a negative number; a warning will return a positive number. `EXGTIM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- no time steps have been stored in the file.

#### **ex\_get\_time: C Interface**

```
int ex_get_time (exoid, time_step, time_value);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`  
The time step number. This is essentially an index (in the time dimension) into the global, nodal, and element variables arrays stored in the database. The first time step is 1.

`void* time_value (W)`  
Returned time at the specified time step.

As an example, the following coding will read the time value stored in the data file for time step `n`:

```
int n, error, exoid;
float time_value;

/* read time value at time step 3 */

n = 3;
error = ex_get_time (exoid, n, &time_value);
```

#### **EXGTIM: Fortran Interface**

```
SUBROUTINE EXGTIM (IDEXO, NSTEP, TIME, IERR)
```

`INTEGER IDEXO (R)`  
EXODUS file ID returned from a previous call to `EXCRE` or `EXOPEN`.

INTEGER NSTEP (R)

The time step number. This is essentially an index (in the time dimension) into the global, nodal, and element variables arrays stored in the database. The first time step is 1.

REAL TIME (W)

Returned time at the specified time step.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

As an example, the following coding will read the time value stored in the data file for time step n:

```
c
c read time value at time step 3
c
      n = 3
      call exgtim (idexo, n, time_value, ierr)
```

### 4.3.7 Read All Time Values

The function `ex_get_all_times` (or `EXGATM` for Fortran) reads the time values for all time steps. Memory must be allocated for the time values array before this function is invoked. The storage requirements (equal to the number of time steps) can be determined by using the `ex_inquire` (or `EXINQ` in Fortran) routine. See Section 4.1.11 on page 41.

Because time values are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_all_times` returns a negative number; a warning will return a positive number. `EXGATM` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- no time steps have been stored in the file.

#### `ex_get_all_times`: C Interface

```
int ex_get_all_times (exoid, time_values);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
void* time_values (W)
```

Returned array of times. These are the time values at all time steps.

The following code segment will read the time values for all time steps stored in the data file:

```
#include "exodusII.h"

int error, exoid, num_time_steps;
float *time_values;

/* determine how many time steps are stored */
error = ex_inquire (exoid, EX_INQ_TIME, &num_time_steps,
                  &fdum, cdum);

/* read time values at all time steps */
time_values = (float *) calloc (num_time_steps, sizeof(float));
error = ex_get_all_times (exoid, time_values);
```

## EXGATM: Fortran Interface

```
SUBROUTINE EXGATM (IDEXO, TIME, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
REAL TIME(*) (W)
```

Returned array of times. These are the time values at all time steps.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following code segment will read the time values for all time steps stored in the data file:

```
c NOTE:      MAXTIM is the maximum number of time steps
c
c      include 'exodusII.inc'
c
c      real time_values(MAXTIM)
c
c determine how many time steps are stored; this can be used if dynamic
c memory allocation is available
c
c      call exinq (idexo, EXTIMS, num_time_steps, fdum, cdum, ierr)
c
c read time values at all time steps
c
c      call exgatm (idexo, time_values, ierr)
```

### 4.3.8 Write Element Variable Truth Table

The function `ex_put_elem_var_tab` (or `EXPVTT` for Fortran) writes the EXODUS II element variable truth table to the database. The element variable truth table indicates whether a particular element result is written for the elements in a particular element block. A 0 (zero) entry indicates that no results will be output for that element variable for that element block. A non-zero entry indicates that the appropriate results will be output.

Although writing the element variable truth table is optional, it is encouraged because it creates at one time all the necessary netCDF variables in which to hold the EXODUS element variable values. This results in significant time savings. See Appendix A for a discussion of efficiency issues.

The function `ex_put_var_param` (or `EXPVP` for Fortran) must be called before this routine in order to define the number of element variables.

In case of an error, `ex_put_elem_var_tab` returns a negative number; a warning will return a positive number. `EXPVTT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- the specified number of element blocks is different than the number specified in a call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_elem_block` (or `EXPELB` for Fortran) not called previously to specify element block parameters.
- `ex_put_var_param` (or `EXPVP` for Fortran) not called previously to specify the number of element variables or was called but with a different number of element variables.
- `ex_put_elem_var` previously called.

#### `ex_put_elem_var_tab`: C Interface

```
int ex_put_elem_var_tab (exoid, num_elem_blk, num_elem_var,  
                        elem_var_tab);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int num\_elem\_blk (R)

The number of element blocks.

int num\_elem\_var (R)

The number of element variables.

```
int elem_var_tab[num_elem_blk,num_elem_var] (R)
```

A 2-dimensional array (with the num\_elem\_var index cycling faster) containing the element variable truth table.

The following coding will create, populate, and write an element variable truth table to an opened EXODUS II file (NOTE: all element variables are valid for all element blocks in this example.):

```
int *truth_tab, num_elem_blk, num_ele_vars, error, exoid;

/* write element variable truth table */
truth_tab = (int *) calloc ((num_elem_blk*num_ele_vars), sizeof(int));

for (i=0, k=0; i<num_elem_blk; i++)
  for (j=0; j<num_ele_vars; j++)
    truth_tab[k++] = 1;

error = ex_put_elem_var_tab (exoid, num_elem_blk, num_ele_vars,
  truth_tab);
```

## EXPVTT: Fortran Interface

```
SUBROUTINE EXPVTT (IDEXO, NELBLK, NVAREL, ISEVOK, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER NELBLK (R)
```

The number of element blocks.

```
INTEGER NVAREL (R)
```

The number of element variables.

```
INTEGER ISEVOK(NVAREL,NELBLK) (R)
```

A 2-dimensional array (with the NVAREL index cycling faster) containing the element variable truth table.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following coding will create, populate, and write an element variable truth table to an opened EXODUS II file. (NOTE: all element variables are valid for all element blocks in this example.):

```
integer truth_tab(num_ele_vars,num_elem_blk)
c
c write element variable truth table
  icnt = 0
  do 30 i = 1,num_elem_blk
    do 20 j = 1,num_ele_vars
      truth_tab(j,i) = 1
20    continue
30    continue
  call expvtt (idexo, num_elem_blk, num_ele_vars, truth_tab, ierr)
```



### 4.3.9 Read Element Variable Truth Table

The function `ex_get_elem_var_tab` (or `EXGVTT` for Fortran) reads the EXODUS II element variable truth table from the database. For a description of the truth table, see the usage of the function `ex_put_elem_var_tab`. Memory must be allocated for the truth table (`num_elem_blk * num_elem_var` in length) before this function is invoked. If the truth table is not stored in the file, it will be created based on information in the file and then returned.

In case of an error, `ex_get_elem_var_tab` returns a negative number; a warning will return a positive number. `EXGVTT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- the specified number of element blocks is different than the number specified in a call to `ex_put_init` (`EXPINI` for Fortran).
- there are no element variables stored in the file or the specified number of element variables doesn't match the number specified in a call to `ex_put_var_param` (or `EXPVP` for Fortran).

#### `ex_get_elem_var_tab`: C Interface

```
int ex_get_elem_var_tab (exoid, num_elem_blk, num_elem_var,  
                        elem_var_tab);
```

```
int exoid (R)
```

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

```
int num_elem_blk (R)
```

The number of element blocks.

```
int num_elem_var (R)
```

The number of element variables.

```
int elem_var_tab[num_elem_blk,num_elem_var] (W)
```

Returned 2-dimensional array (with the `num_elem_var` index cycling faster) containing the element variable truth table.

As an example, the following coding will read the element variable truth table from an opened EXODUS II file:

```
int *truth_tab, num_elem_blk, num_ele_vars, error, exoid;  
  
truth_tab = (int *) calloc ((num_elem_blk*num_ele_vars), sizeof(int));  
  
error = ex_get_elem_var_tab (exoid, num_elem_blk, num_ele_vars,  
                            truth_tab);
```

## EXGVTT: Fortran Interface

```
SUBROUTINE EXGVTT (IDEXO, NELBLK, NVAREL, ISEVOK, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER NELBLK (R)
```

The number of element blocks.

```
INTEGER NVAREL (R)
```

The number of element variables.

```
INTEGER ISEVOK(NVAREL, NELBLK) (W)
```

Returned 2-dimensional array (with the NVAREL index cycling faster) containing the element variable truth table.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

As an example, the following coding will read the element variable truth table from an opened EXODUS II file:

```
integer truth_tab(num_ele_vars,num_elem_blk)
c
c read element variable truth table
c
call exgvtt (idexo, num_elem_blk, num_ele_vars, truth_tab, ierr)
```

### 4.3.10 Write Element Variable Values at a Time Step

The function `ex_put_elem_var` (or `EXPEV` for Fortran) writes the values of a single element variable for one element block at one time step. It is recommended, but not required, to write the element variable truth table (with `ex_put_elem_var_tab` for C; `EXPVTT` for Fortran) before this function is invoked for better efficiency. See Appendix A for a discussion of efficiency issues.

Because element variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_elem_var` returns a negative number; a warning will return a positive number. `EXPEV` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- invalid element block ID.
- `ex_put_elem_block` (or `EXPELB` for Fortran) not called previously to specify parameters for this element block.
- `ex_put_var_param` (or `EXPVP` for Fortran) not called previously specifying the number of element variables.
- an element variable truth table was stored in the file but contains a zero (indicating no valid element variable) for the specified element block and element variable.

#### **ex\_put\_elem\_var: C Interface**

```
int ex_put_elem_var (exoid, time_step, elem_var_index,  
                    elem_blk_id, num_elem_this_blk, elem_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`

The time step number, as described under `ex_put_time`. This is essentially a counter that is incremented only when results variables are output. The first time step is 1.

`int elem_var_index (R)`

The index of the element variable. The first variable has an index of 1.

`int elem_blk_id (R)`

The element block ID.

`int num_elem_this_blk (R)`

The number of elements in the given element block.

```
void* elem_var_vals (R)
    Array of num_elem_this_blk values of the elem_var_indeth element variable
    for the element block with ID of elem_blk_id at the time_stepth time step.
```

The following coding will write out all of the element variables for a single time step n to an open EXODUS II file:

```
int num_ele_vars, num_elem_blk, *num_elem_in_block,error, exoid, n,
    *ebids;
float *elem_var_vals;

/* write element variables */

for (k=1; k<=num_ele_vars; k++)
{
    for (j=0; j<num_elem_blk; j++)
    {
        elem_var_vals = (float *)
            calloc (num_elem_in_block[j], sizeof(float));

        for (m=0; m<num_elem_in_block[j]; m++)
        {
            /* simulation code fills this in */
            elem_var_vals[m] = 10.0;
        }
        error = ex_put_elem_var (exoid, n, k, ebids[j],
            num_elem_in_block[j], elem_var_vals);
        free (elem_var_vals);
    }
}
```

## EXPEV: Fortran Interface

```
SUBROUTINE EXPEV (IDEXO, ISTEP, IXELEV, IDELB, NUMELB,
    VALEV, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER ISTEP (R)
```

The time step number, as described under EXPTIM. This is essentially a counter that is incremented only when results variables are output. The first time step is 1.

```
INTEGER IXELEV (R)
```

The index of the element variable. The first variable has an index of 1.

```
INTEGER IDELB (R)
```

The element block ID.

```
INTEGER NUMELB (R)
```

The number of elements in the given element block.

REAL VALEV(\*) (R)

Array of NUMELB values of the IXELEVth element variable for the element block with ID of IDELB at the ISTEPth time step.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

The following coding will write out all of the element variables for a single time step n to an open EXODUS II file:

```
c NOTE:      MAXEBK is maximum number of element blocks
c           MAXELB is maximum number of elements per block
c
c           integer num_elem_in_block(MAXEBK)
c           real elem_var_vals(MAXELB)
c
c write element variables
c
c           do 100 k = 1, num_ele_vars
c             do 90 j = 1, num_elem_blk
c               do 80 m = 1, num_elem_in_block(j)
c
c analysis code fills this array
c
c               elem_var_vals(m) = 10.0
c
c             80 continue
c
c               call expev (idexo, n, k, num_elem_in_block(j),
c                 1      elem_var_vals, ierr)
c
c           90 continue
c         100 continue
```

### 4.3.11 Read Element Variable Values at a Time Step

The function `ex_get_elem_var` (or `EXGEV` for Fortran) reads the values of a single element variable for one element block at one time step. Memory must be allocated for the element variable values array before this function is invoked.

Because element variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_elem_var` returns a negative number; a warning will return a positive number. `EXGEV` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- variable does not exist for the desired element block.
- invalid element block.

#### `ex_get_elem_var`: C Interface

```
int ex_get_elem_var (exoid, time_step, elem_var_index,  
                    elem_blk_id, num_elem_this_blk, elem_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`

The time step number, as described under `ex_put_time`, at which the element variable values are desired. This is essentially an index (in the time dimension) into the element variable values array stored in the database. The first time step is 1.

`int elem_var_index (R)`

The index of the desired element variable. The first variable has an index of 1.

`int elem_blk_id (R)`

The desired element block ID.

`int num_elem_this_blk (R)`

The number of elements in this element block.

`void* elem_var_vals (W)`

Returned array of `num_elem_this_blk` values of the `elem_var_index`th element variable for the element block with ID of `elem_blk_id` at the `time_step`th time step.

As an example, the following code segment will read the `var_index`th element variable at one time step stored in an EXODUS II file:

```
int *ids, num_elem_blk, error, exoid, *num_elem_in_block, step, var_ind;  
float *var_vals;
```

```

ids = (int *) calloc(num_elem_blk, sizeof(int));
error = ex_get_elem_blk_ids (exoid, ids);

step = 1; /* read at the first time step */
for (i=0; i<num_elem_blk; i++) {
    var_vals = (float *) calloc (num_elem_in_block[i], sizeof(float));
    error = ex_get_elem_var (exoid, step, var_ind, ids[i],
        num_elem_in_block[i], var_vals);
    free (var_values); }

```

## EXGEV: Fortran Interface

```

SUBROUTINE EXGEV (IDEXO, ISTEP, IXELEV, IDELB, NUMELB,
    VALEV, IERR)

```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER ISTEP (R)

The time step number, as described under EXPTIM, at which the element variable is desired. This is essentially an index (in the time dimension) into the element variable values array stored in the database. The first time step is 1.

INTEGER IXELEV (R)

The index of the desired element variable. The first variable has an index of 1.

INTEGER IDELB (R)

The desired element block ID.

INTEGER NUMELB (R)

The number of elements in this element block.

REAL VALEV(\*) (W)

Returned array of NUMELB values of the IXELEVth element variable for the element block with ID of IDELB at the ISTEPth time step.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

As an example, the following code segment will read the var\_indexth element variable at one time step stored in an EXODUS II file:

```

c NOTE:      MAXEBK is maximum number of element blocks
c           MAXELB is maximum number of elements per block
integer ids(MAXEBK), var_index, num_elem_in_block(MAXEBK)
real var_values(MAXELB)

call exgebi (idexo, ids, ierr)

do 10 i = 1, num_elem_blk
    call exgev (idexo, istep, var_index, ids(i),
1        num_elem_in_block(i), var_values, ierr)
10    continue

```

### 4.3.12 Read Element Variable Values through Time

The function `ex_get_elem_var_time` (or `EXGEVT` for Fortran) reads the values of an element variable for a single element through a specified number of time steps. Memory must be allocated for the element variable values array before this function is invoked.

Because element variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_elem_var_time` returns a negative number; a warning will return a positive number. `EXGEVT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_elem_block` (or `EXPELB` for Fortran) not called previously to specify parameters for all element blocks.
- variable does not exist for the desired element or results haven’t been written.

#### `ex_get_elem_var_time`: C Interface

```
int ex_get_elem_var_time (exoid, int elem_var_index, int
    elem_number, int beg_time_step, int end_time_step,
    elem_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_var_index (R)`

The index of the desired element variable. The first variable has an index of 1.

`int elem_number (R)`

The internal ID (see Section 3.5 on page 7) of the desired element. The first element is 1.

`int beg_time_step (R)`

The beginning time step for which an element variable value is desired. This is not a time value but rather a time step number, as described under `ex_put_time`. The first time step is 1.

`int end_time_step (R)`

The last time step for which an element variable value is desired. If negative, the last time step in the database will be used. The first time step is 1.

`void* elem_var_vals (W)`

Returned array of  $(end\_time\_step - beg\_time\_step + 1)$  values of the  $elem\_number$ th element for the  $elem\_var\_index$ th element variable.



For example, the following coding will read the values of the `var_index`th element variable for element number 2 from the first time step to the last time step:

```
#include "exodusII.h"

int error, exoid, num_time_steps, var_index, elem_num, beg_time,
    end_time;
float *var_values;

/* determine how many time steps are stored */
error = ex_inquire (exoid, EX_INQ_TIME, &num_time_steps, &fdum, cdum);

/* read an element variable through time */
var_values = (float *) calloc (num_time_steps, sizeof(float));

var_index = 2;
elem_num = 2;
beg_time = 1;
end_time = -1;

error = ex_get_elem_var_time (exoid, var_index, elem_num,
    beg_time, end_time, var_values);
```

## EXGEVT: Fortran Interface

```
SUBROUTINE EXGEVT (IDEXO, IXELEV, IELNUM, ISTEPB, ISTPE,
    VALEV, IERR)
```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IXELEV (R)

The index of the desired element variable. The first variable has an index of 1.

INTEGER IELNUM (R)

The internal ID (see Section 3.5 on page 7) of the desired element. The first element is 1.

INTEGER ISTEPB (R)

The beginning time step for which an element variable value is desired. This is not a time value but rather a time step number, as described under EXPTIM. The first time step is 1.

INTEGER ISTPE (R)

The last time step for which an element variable value is desired. If negative, the last time step in the database will be used. The first time step is 1.

REAL VALEV(\*) (W)

Returned array of (ISTPE - ISTEPB + 1) values of the IELNUMth element for the IXELEVth element variable.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For example, the following coding will read the values of the var\_indexth element variable for element number 2 from the first time step to the last time step:

```
c NOTE:  MAXVAL is the maximum number of values to be read
c
      integer var_index, elem_num, beg_time, end_time
      real var_values(MAXVAL)
c
c read an element variable through time
c
      var_index = 2
      elem_num = 2
      beg_time = 1
      end_time = -1

      call exgevt (idexo, var_index, elem_num, beg_time, end_time,
1  var_values, ierr)
```

### 4.3.13 Write Global Variables Values at a Time Step

The function `ex_put_glob_vars` (or `EXPGV` for Fortran) writes the values of all the global variables for a single time step. The function `ex_put_var_param` (`EXPVP` for Fortran) must be invoked before this call is made.

Because global variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_glob_vars` returns a negative number; a warning will return a positive number. `EXPGV` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- `ex_put_var_param` (or `EXPVP` for Fortran) not called previously specifying the number of global variables.

#### `ex_put_glob_vars`: C Interface

```
int ex_put_glob_vars (exoid, time_step, num_glob_vars,  
                    glob_var_vals);
```

`int exoid (R)`  
EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`  
The time step number, as described under `ex_put_time`. This is essentially a counter that is incremented when results variables are output. The first time step is 1.

`int num_glob_vars (R)`  
The number of global variables to be written to the database.

`void* glob_var_vals (R)`  
Array of `num_glob_vars` global variable values for the `time_step`th time step.

As an example, the following coding will write the values of all the global variables at one time step to an open EXODUS II file:

```
int num_glo_vars, error, exoid, time_step;  
float *glob_var_vals  
  
/* write global variables */  
for (j=0; j<num_glo_vars; j++)  
    /* application code fills this array */  
    glob_var_vals[j] = 10.0;  
  
error = ex_put_glob_vars (exoid, time_step, num_glo_vars,  
                        glob_var_vals);
```

## EXPGV: Fortran Interface

```
SUBROUTINE EXPGV (IDEXO, ISTEP, NVARGL, VALGV, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER ISTEP (R)
```

The time step number, as described under EXPTIM. This is essentially a counter that is incremented only when results variables are output. The first time step is 1.

```
INTEGER NVARGL (R)
```

The number of global variables to be written to the database.

```
REAL VALGV(*) (R)
```

Array of NVARGL global variable values for the ISTEPth time step.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

As an example, the following coding will write the values of all the global variables at one time step to an open EXODUS II file:

```
c NOTE: MAXGVAR is the maximum number of global variables
c
      integer num_glo_vars
      real glob_var_vals(MAXGVAR)
c
c write all global variables for time step istep
c
      do 50 j = 1, num_glo_vars
c
c application code fills in this array
c
          glob_var_vals(j) = 10.0
50      continue

      call expgv (idexo, istep, num_glo_vars, glob_var_vals, ierr)
```

### 4.3.14 Read Global Variables Values at a Time Step

The function `ex_get_glob_vars` (or `EXGGV` for Fortran) reads the values of all the global variables for a single time step. Memory must be allocated for the global variables values array before this function is invoked.

Because global variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_glob_vars` returns a negative number; a warning will return a positive number. `EXGGV` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- no global variables stored in the file.
- a warning value is returned if no global variables are stored in the file.

#### **ex\_get\_glob\_vars: C Interface**

```
int ex_get_glob_vars (exoid, time_step, num_glob_vars,  
                    glob_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`

The time step, as described under `ex_put_time`, at which the global variable values are desired. This is essentially an index (in the time dimension) into the global variable values array stored in the database. The first time step is 1.

`int num_glob_vars (R)`

The number of global variables stored in the database.

`void* glob_var_vals (W)`

Returned array of `num_glob_vars` global variable values for the `time_step`th time step.

The following is an example code segment that reads all the global variables at one time step:

```
int num_glo_vars, error, time_step;  
float *var_values;  
  
error = ex_get_var_param (idexo, "g", &num_glo_vars);  
var_values = (float *) calloc (num_glo_vars, sizeof(float));  
error = ex_get_glob_vars (idexo, time_step, num_glo_vars, var_values);
```

## EXGGV: Fortran Interface

```
SUBROUTINE EXGGV (IDEXO, ISTEP, NVARGL, VALGV, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER ISTEP (R)
```

The time step number, as described under EXPTIM, at which global variables are desired. This is essentially an index (in the time dimension) into the global variable values array stored in the database. The first time step is 1.

```
INTEGER NVARGL (R)
```

The number of global variables stored in the database.

```
REAL VALGV(*) (W)
```

Returned array of NVARGL global variable values for the ISTEPth time step.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

The following is an example code segment that reads all the global variables at one time step:

```
c NOTE: MAXGVAR is the maximum number of global variables
c
c     real var_values(MAXGVAR)
c
c read all global variables at one time step
c
c     call exggv (idexo, istep, num_glo_vars, var_values, ierr)
```

### 4.3.15 Read Global Variable Values through Time

The function `ex_get_glob_var_time` (or `EXGGVT` for Fortran) reads the values of a single global variable through a specified number of time steps. Memory must be allocated for the global variable values array before this function is invoked.

Because global variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_glob_var_time` returns a negative number; a warning will return a positive number. `EXGGVT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- specified global variable does not exist.
- a warning value is returned if no global variables are stored in the file.

#### `ex_get_glob_var_time`: C Interface

```
int ex_get_glob_var_time (exoid, glob_var_index,  
    beg_time_step, end_time_step, glob_var_vals);
```

int exoid (R)

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

int glob\_var\_index (R)

The index of the desired global variable. The first variable has an index of 1.

int beg\_time\_step (R)

The beginning time step for which a global variable value is desired. This is not a time value but rather a time step number, as described under `ex_put_time`. The first time step is 1.

int end\_time\_step (R)

The last time step for which a global variable value is desired. If negative, the last time step in the database will be used. The first time step is 1.

void\* glob\_var\_vals (W)

Returned array of  $(end\_time\_step - beg\_time\_step + 1)$  values for the `glob_var_index`th global variable.

The following is an example of using this function:

```
#include "exodusII.h"  
int error, exoid, num_time_steps, var_index, beg_time, end_time;  
float *var_values;  
  
/* determine how many time steps are stored */  
error = ex_inquire (exoid, EX_INQ_TIME, &num_time_steps, &fdum, cdum);
```

```

/* read the first global variable for all time steps */

var_index = 1;
beg_time = 1;
end_time = -1;

var_values = (float *) calloc (num_time_steps, sizeof(float));

error = ex_get_glob_var_time (exoid, var_index, beg_time, end_time,
    var_values);

```

## EXGGVT: Fortran Interface

```

SUBROUTINE EXGGVT (IDEXO, IXGLOV, ISTEPB, ISTPE, VALGV, IERR)

```

```

INTEGER IDEXO (R)

```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```

INTEGER IXGLOV (R)

```

The index of the desired global variable. The first variable has an index of 1.

```

INTEGER ISTEPB (R)

```

The beginning time step for which a global variable value is desired. This is not a time value but rather a time step number, as described under EXPTIM. The first time step is 1.

```

INTEGER ISTPE (R)

```

The last time step for which a global variable value is desired. If negative, the last time step in the database will be used. The first time step is 1.

```

REAL VALGV(*) (W)

```

Returned array of (ISTPE - ISTEPB + 1) values for the IXGLOVth global variable.

```

INTEGER IERR (W)

```

Returned error code. If no errors occurred, 0 is returned.

The following is an example of using this function:

```

c NOTE: MAXVAL is the maximum number of values to be read
c
c     integer var_index, beg_time, end_time
c     real var_values(MAXVAL)
c
c read a single global variable for all time steps
c
c     var_index = 1
c     beg_time = 1
c     end_time = -1
c     call exggvt (idexo, var_index, beg_time, end_time, var_values, ierr)

```



### 4.3.16 Write Nodal Variable Values at a Time Step

The function `ex_put_nodal_var` (or `EXPNV` for Fortran) writes the values of a single nodal variable for a single time step. The function `ex_put_var_param` (`EXPVP` for Fortran) must be invoked before this call is made.

Because nodal variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_put_nodal_var` returns a negative number; a warning will return a positive number. `EXPNV` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- data file opened for read only.
- data file not initialized properly with call to `ex_put_init` (`EXPINI` for Fortran).
- `ex_put_var_param` (or `EXPVP` for Fortran) not called previously specifying the number of nodal variables.

#### `ex_put_nodal_var`: C Interface

```
int ex_put_nodal_var (exoid, time_step, nodal_var_index,  
                    num_nodes, nodal_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`

The time step number, as described under `ex_put_time`. This is essentially a counter that is incremented when results variables are output. The first time step is 1.

`int nodal_var_index (R)`

The index of the nodal variable. The first variable has an index of 1.

`int num_nodes (R)`

The number of nodal points.

`void* nodal_var_vals (R)`

Array of `num_nodes` values of the `nodal_var_index`th nodal variable for the `time_step`th time step.

As an example, the following code segment writes all the nodal variables for a single time step:

```
int num_nod_vars, num_nodes, error, exoid, time_step;  
float *nodal_var_vals;
```

```

/* write nodal variables */

nodal_var_vals = (float *) calloc (num_nodes, sizeof(float));

for (k=1; k<=num_nod_vars; k++)
{
  for (j=0; j<num_nodes; j++)
    /* application code fills in this array */
    nodal_var_vals[j] = 10.0;

  error = ex_put_nodal_var (exoid, time_step, k, num_nodes,
    nodal_var_vals);
}

```

## EXPV: Fortran Interface

```
SUBROUTINE EXPV (IDEXO, ISTEP, IXNODV, NUMNP, VALNV, IERR)
```

```
INTEGER IDEXO (R)
```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```
INTEGER ISTEP (R)
```

The time step number, as described under EXPTIM. This is essentially a counter that is incremented when results variables are output. The first time step is 1.

```
INTEGER IXNODV (R)
```

The index of the nodal variable. The first variable has an index of 1.

```
INTEGER NUMNP (R)
```

The number of nodal points.

```
REAL VALNV(*) (R)
```

Array of NUMNP values of the IXNODVth nodal variable for the ISTEPth time step.

```
INTEGER IERR (W)
```

Returned error code. If no errors occurred, 0 is returned.

As an example, the following code segment writes all the nodal variables for a single time step:

```

      real nodal_var_vals(MAXNOD)

      do 70 k = 1, num_nod_vars
        do 60 j = 1, num_nodes
c          simulation code fills in this array
          nodal_var_vals(j) = 10.0
60        continue

          call expv (idexo, istep, k, num_nodes, nodal_var_vals, ierr)
70      continue

```

### 4.3.17 Read Nodal Variable Values at a Time Step

The function `ex_get_nodal_var` (or `EXGNV` for Fortran) reads the values of a single nodal variable for a single time step. Memory must be allocated for the nodal variable values array before this function is invoked.

Because nodal variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_nodal_var` returns a negative number; a warning will return a positive number. `EXGNV` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- data file not properly opened with call to `ex_create` or `ex_open` (`EXCRE` or `EXOPEN` for Fortran).
- specified nodal variable does not exist.
- a warning value is returned if no nodal variables are stored in the file.

#### **ex\_get\_nodal\_var: C Interface**

```
int ex_get_nodal_var (exoid, int time_step, nodal_var_index,  
                    num_nodes, nodal_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step (R)`

The time step, as described under `ex_put_time`, at which the nodal variable values are desired. This is essentially an index (in the time dimension) into the nodal variable values array stored in the database. The first time step is 1.

`int nodal_var_index (R)`

The index of the desired nodal variable. The first variable has an index of 1.

`int num_nodes (R)`

The number of nodal points.

`void* nodal_var_vals (W)`

Returned array of `num_nodes` values of the `nodal_var_index`th nodal variable for the `time_step`th time step.

For example, the following demonstrates how this function would be used:

```
int num_nodes, time_step, var_index;  
float *var_values;  
  
/* read the second nodal variable at the first time step */  
time_step = 1;  
var_index = 2;
```

```

var_values = (float *) calloc (num_nodes, sizeof(float));

error = ex_get_nodal_var (exoid, time_step, var_index, num_nodes,
var_values);

```

## EXGNV: Fortran Interface

```

SUBROUTINE EXGNV (IDEXO, ISTEP, IXNODV, NUMNP, VALNV, IERR)

```

```

INTEGER IDEXO (R)

```

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

```

INTEGER ISTEP (R)

```

The time step number, as described under EXPTIM, at which the nodal variable is desired. This is essentially an index (in the time dimension) into the nodal variable values array stored in the database. The first time step is 1.

```

INTEGER IXNODV (R)

```

The index of the desired nodal variable. The first variable has an index of 1.

```

INTEGER NUMNP (R)

```

The number of nodal points.

```

REAL VALNV(*) (W)

```

Returned array of NUMNP values of the IXNODVth nodal variable for the ISTEPth time step.

```

INTEGER IERR (W)

```

Returned error code. If no errors occurred, 0 is returned.

For example, the following demonstrates how this function would be used:

```

c NOTE: MAXNOD is the maximum number of nodes for the model
c
c       integer var_index
c       real var_values(MAXNOD)
c
c read a nodal variable at one time step
c
c       istep = 10
c       var_index = 2
c       num_nodes = 1000
c
c       call exgnv (idexo, istep, var_index, num_nodes, var_values, ierr)

```

### 4.3.18 Read Nodal Variable Values through Time

The function `ex_get_nodal_var_time` (or `EXGNVT` for Fortran) reads the values of a nodal variable for a single node through a specified number of time steps. Memory must be allocated for the nodal variable values array before this function is invoked.

Because nodal variables are floating point values, the application code must declare the array passed to be the appropriate type (“float” or “double” in C; “REAL\*4” or “REAL\*8” in Fortran) to match the compute word size passed in `ex_create` (or `EXCRE` for Fortran) or `ex_open` (or `EXOPEN` for Fortran).

In case of an error, `ex_get_nodal_var_time` returns a negative number; a warning will return a positive number. `EXGNVT` returns a nonzero error (negative) or warning (positive) number in `IERR`. Possible causes of errors include:

- specified nodal variable does not exist.
- a warning value is returned if no nodal variables are stored in the file.

#### `ex_get_nodal_var_time`: C Interface

```
int ex_get_nodal_var_time (exoid, nodal_var_index,  
    node_number, beg_time_step, end_time_step,  
    nodal_var_vals);
```

`int exoid (R)`

EXODUS file ID returned from a previous call to `ex_create` or `ex_open`.

`int nodal_var_index (R)`

The index of the desired nodal variable. The first variable has an index of 1.

`int node_number (R)`

The internal ID (see Section 3.5 on page 7) of the desired node. The first node is 1.

`int beg_time_step (R)`

The beginning time step for which a nodal variable value is desired. This is not a time value but rather a time step number, as described under `ex_put_time`. The first time step is 1.

`int end_time_step (R)`

The last time step for which a nodal variable value is desired. If negative, the last time step in the database will be used. The first time step is 1.

`void* nodal_var_vals (W)`

Returned array of  $(end\_time\_step - beg\_time\_step + 1)$  values of the  $node\_number$ th node for the  $nodal\_var\_index$ th nodal variable.

For example, the following code segment will read the values of the first nodal variable for node number one for all time steps stored in the data file:

```
#include "exodusII.h"  
int num_time_steps, var_index, node_num, beg_time, end_time, error,  
    exoid;
```

```

float *var_values;

/* determine how many time steps are stored */
error = ex_inquire (exoid, EX_INQ_TIME, &num_time_steps, &fdum, cdum);

/* read a nodal variable through time */
var_values = (float *) calloc (num_time_steps, sizeof(float));

var_index = 1; node_num = 1; beg_time = 1; end_time = -1;
error = ex_get_nodal_var_time (exoid, var_index, node_num, beg_time,
    end_time, var_values);

```

## EXGNVT: Fortran Interface

```

SUBROUTINE EXGNVT (IDEXO, IXNODV, NODNUM, ISTEPB, ISTPE,
    VALNV, IERR)

```

INTEGER IDEXO (R)

EXODUS file ID returned from a previous call to EXCRE or EXOPEN.

INTEGER IXNODV (R)

The index of the desired nodal variable. The first variable has an index of 1.

INTEGER NODNUM (R)

The internal ID (see Section 3.5 on page 7) of the desired node. The first node is 1.

INTEGER ISTEPB (R)

The beginning time step for which a nodal variable value is desired. This is not a time value but rather a time step number, as described under EXPTIM. The first time step is 1.

INTEGER ISTPE (R)

The last time step for which a nodal variable value is desired. If negative, the last time step in the database will be used. The first time step is 1.

REAL VALNV(\*) (W)

Returned array of (ISTPE - ISTEPB + 1) values of the NODNUMth node for the IXNODVth nodal variable.

INTEGER IERR (W)

Returned error code. If no errors occurred, 0 is returned.

For example, the following code segment will read the values of the first nodal variable for node number one for all time steps stored in the data file:

```

    integer var_ind, btime, etime
    real var_vals(MAXVAL)
c
c read a nodal variable through time
c
    var_ind = 1
    node_num = 1
    btime = 1
    etime = -1
    call exgnvt (idexo, var_ind, node_num, btime, etime, var_vals, ierr)

```

## 5 References

- [1] W. C. Mills-Curran, A. P. Gilkey, and D. P. Flanagan, "EXODUS: A Finite Element File Format for Pre- and Post-processing," Technical Report SAND87-2977, Sandia National Laboratories, Albuquerque, New Mexico, September 1988.
- [2] G. D. Sjaardema, "Overview of the Sandia National Laboratories Engineering Analysis Code Access System," Technical Report SAND92-2292, Sandia National Laboratories, Albuquerque, New Mexico, January 1993.
- [3] R. K. Rew, G. P. Davis, and S. Emmerson, "NetCDF User's Guide: An Interface for Data Access," Version 2.3, University Corporation for Atmospheric Research, Boulder, Colorado, April 1993.
- [4] Sun Microsystems, "External Data Representation Standard: Protocol Specification," RFC 1014; Information Sciences Institute, May 1988.
- [5] PDA Engineering, "PATRAN Plus User Manual," Publication No. 2191024, Costa Mesa, California, January 1990.

Intentionally Left Blank



# Appendix A

## Implementation of EXODUS II with netCDF

### Description

The netCDF software is an I/O library, callable from C or Fortran, which stores and retrieves scientific data structures in self-describing, machine-independent files. “Self-describing” means that a file includes information defining the data it contains. “Machine-independent” means that a file is represented in a form that can be accessed by computers with different ways of storing integers, characters, and floating-point numbers. It is available via anonymous FTP from [unidata.ucar.edu](http://unidata.ucar.edu) in the file `pub/netcdf/netcdf.tar.Z`.

For the EXODUS II implementation, the standard netCDF distribution is used except that the following defined constants in the include file `netcdf.h` are modified to the values shown:

```
#define MAX_NC_DIMS 8192
#define MAX_NC_ATTRS 1024
#define MAX_NC_VARS 8192
#define MAX_NC_NAME 256
```

### Efficiency Issues

There are some efficiency concerns with using netCDF as the low level data handler. The main one is that whenever a new object is introduced, the file is put into “define” mode, the new object is defined, and then the file is taken out of “define” mode. A result of going in and out of “define” mode is that all of the data that was output previous to the introduction of the new object is copied to a new file. Obviously, this copying of data to a new file is very inefficient. We have attempted to minimize the number of times the data file is put into “define” mode by accumulating objects within a single EXODUS II API function. Thus using optional features such as the element variable truth table, concatenated node and side sets, and writing all property array names with `ex_put_prop_names` (EXPPN for Fortran) will increase efficiency significantly.

### netCDF Data Objects

This section describes how EXODUS II data are mapped to netCDF entities. This information is needed only for those individuals who desire to access an EXODUS II

database via netCDF calls directly or desire to modify the routines that comprise the Application Programming Interface (API).

The following is a list of the names of the data entities found in an EXODUS II file and a description of each entity. The names are constants predefined in the include file `exodusII_int.h` for C or `exodusII_int.inc` for Fortran. They are grouped into three netCDF categories: attributes, dimensions, and variables.

### **Attributes**

An attribute is used to describe data entities. It can be global (describe entire file) or attached to a dimension or variable.

- 1) `title`                    the database title; character global attribute
- 2) `version`                the EXODUS II file version number; float global attribute
- 3) `api version`            the EXODUS II API version number; float global attribute
- 4) `floating point word size`  
word size of floating point numbers in the file; int global attribute
- 5) `elem_type`              element type names for each element block; character variable attribute attached to `connect` variable
- 6) `name`                    name of element block, node set, or side set property; character variable attribute attached to specific property

### **Dimensions**

A dimension is an integer scalar value that is used to define the size of variables.

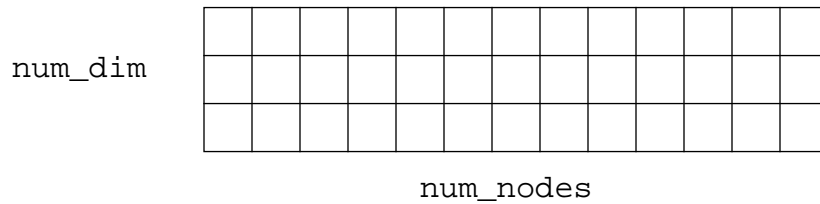
- 1) `num_nodes`              number of nodes
- 2) `num_dim`                number of dimensions of the finite element model; 1-, 2-, or 3-d
- 3) `num_elem`                number of elements
- 4) `num_el_blk`              number of element blocks
- 5) `num_el_in_blk#`        number of elements in element block #
- 6) `num_nod_per_el#`        number of nodes per element in element block #
- 7) `num_att_in_blk#`        number of attributes per element in element block #

8) num_side_sets	number of side sets
9) num_side_ss#	number of sides (also the number of elements) in side set #
10) num_df_ss#	number of distribution factors in side set #
11) num_node_sets	number of node sets
12) num_nod_ns#	number of nodes in node set #
13) num_df_ns#	number of distribution factors in node set #
14) num_qa_rec	number of QA records
15) num_info	number of information records
16) num_glo_var	number of global variables
17) num_nod_var	number of nodal variables
18) num_elem_var	number of element variables
19) time_step	unlimited (expandable) dimension for time steps
20) len_string	length of a string; currently set to allow 32 characters (plus NULL character for C interface)
21) len_line	length of a line; currently set to allow 80 characters (plus NULL character for C interface)
22) four	number of strings in a single QA record

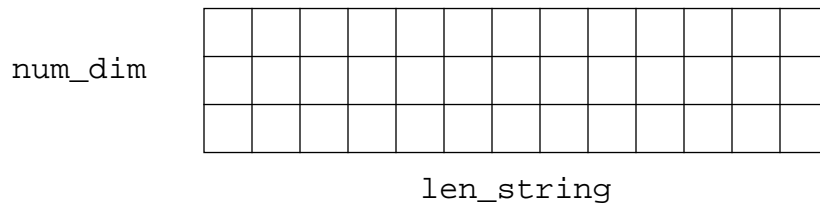
## Variables

A variable is an entity that contains data. Its size and shape are specified by dimensions. Note that the order of the dimensions is “row order” as implemented in the C language, so the last dimension specified varies fastest, the first dimension varies slowest. For multi-dimension variables, illustrations are included in the descriptions below for ease of understanding. For variables that are dimensioned through time, ellipses (. . .) are used to show that the variable can expand in that dimension.

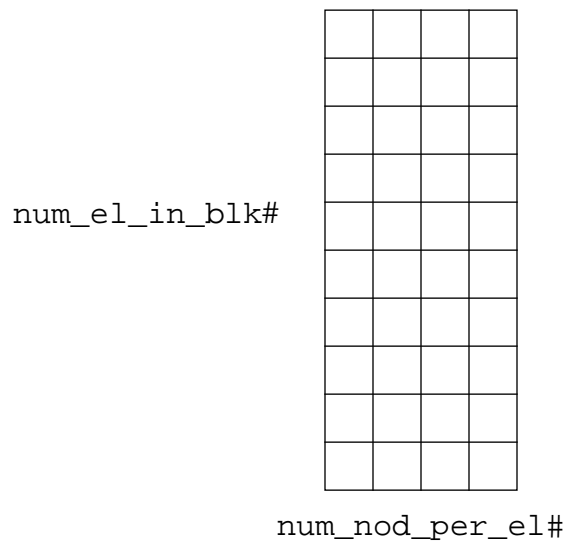
- 1) `coord (num_dim, num_nodes)`  
nodal coordinates; float or double



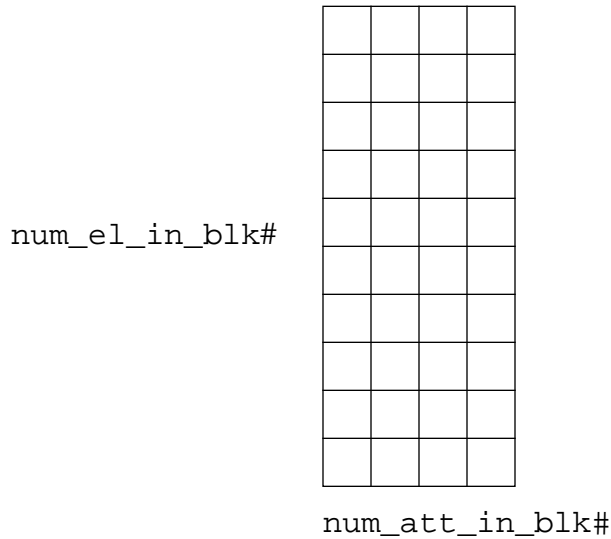
- 2) `coor_names (num_dim, len_string)`  
names of coordinates; character



- 3) `connect# (num_el_in_blk#, num_nod_per_el#)`  
element connectivity for element block #; integer



4) attrib# (num\_el\_in\_blk#, num\_att\_in\_blk#)  
list of attributes for element block #; float or double



5) eb\_prop# (num\_el\_blk)  
list of the #th property for all element blocks; integer

6) elem\_map (num\_elem)  
element order map; integer

7) dist\_fact\_ss# (num\_df\_ss#)  
distribution factors for each node in side set #; float or double

8) elem\_ss# (num\_side\_ss#)  
list of elements in side set #; integer

9) side\_ss# (num\_side\_ss#)  
list of sides in side set #; integer

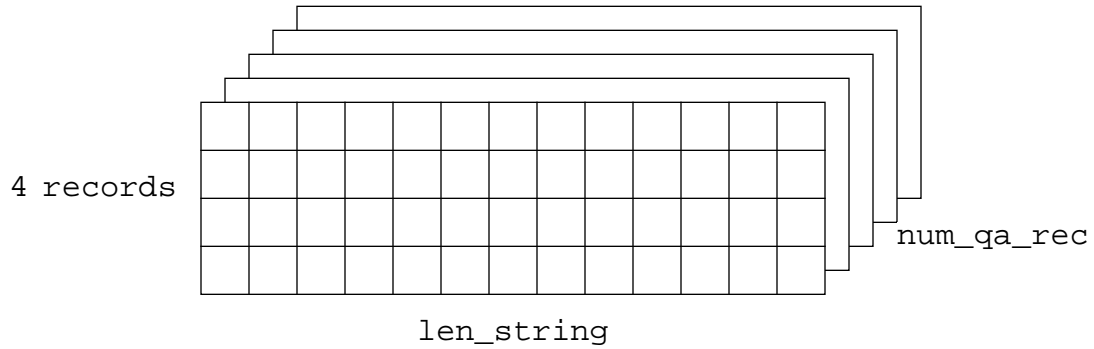
10) ss\_prop# (num\_side\_sets)  
list of the #th property for all side sets; integer

11) node\_ns# (num\_nod\_ns#)  
list of nodes in node set #; integer

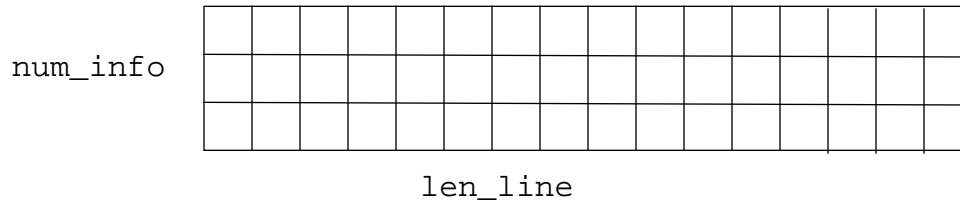
12) dist\_fact\_ns# (num\_nod\_ns#)  
list of distribution factors in node set #; float or double

13) ns\_prop# (num\_node\_sets)  
list of the #th property for all node sets; integer

14) qa\_records (num\_qa\_rec, 4, len\_string)  
QA records; character

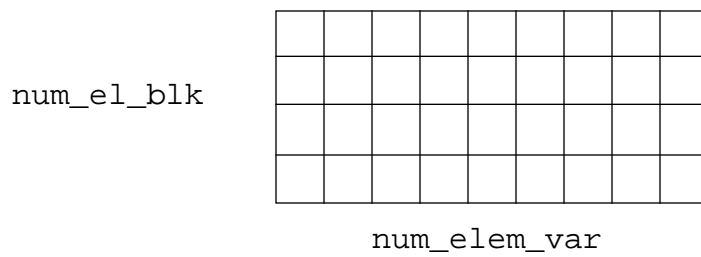


15) info\_records (num\_info, len\_line)  
information records; character



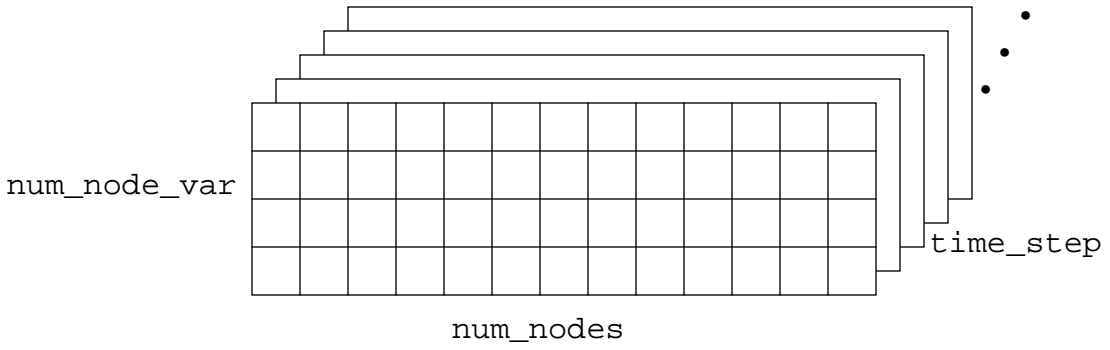
16) time\_whole (time\_step)  
simulation times for time steps; float or double

17) elem\_var\_tab (num\_el\_blk, num\_elem\_var)  
element variable truth table; integer

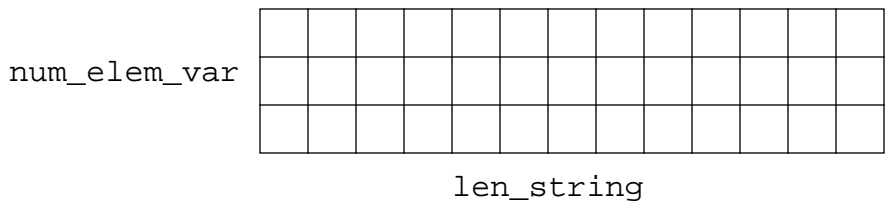




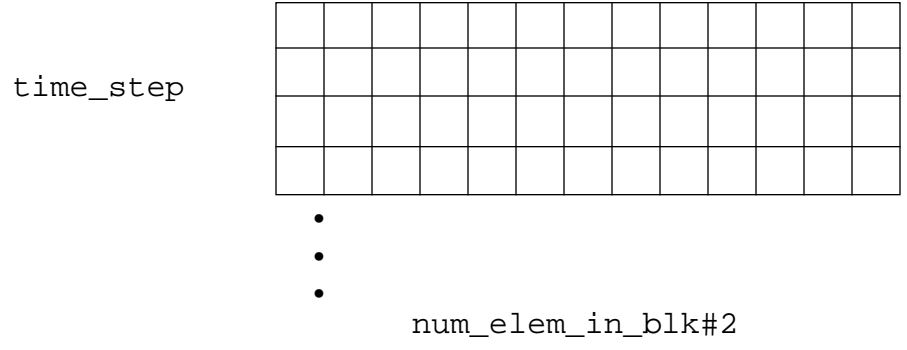
21) vals\_nod\_var (time\_step, num\_nod\_var, num\_nodes)  
 values of nodal variables; float or double



22) name\_elem\_var (num\_elem\_var, len\_string)  
 names of element variables; character



23) vals\_elem\_var#1eb#2 (time\_step, num\_el\_in\_blk#2)  
 values of element variable #1 in element block #2; for each element block, there is one of these for each element variable that is valid for that element block; float or double





# Appendix B

## Function Call Summary

This appendix includes an alphabetized list of EXODUS II functions, passed arguments, and page number where their descriptions are located in the manual. The C interface routines are listed first followed by the FORTRAN binding routines.

### C binding routines

```
int ex_close (
    int exoid);                               page 27

int ex_create (
    char* path,
    int cmode,
    int* comp_ws,
    int* io_ws);                               page 23

int ex_cvt_nodes_to_sides (
    int exoid,
    int* num_side_per_set,
    int* num_nodes_per_set,
    int* side_sets_elem_index,
    int* side_sets_node_index,
    int* side_sets_elem_list,
    int* side_sets_node_list,
    int* side_sets_side_list);               page 115

void ex_err (
    char* module_name,
    char* message,
    int err_num);                             page 45

int ex_get_all_times (
    int exoid,
    void* time_values);                       page 143

int ex_get_concat_node_sets (
    int exoid,
    int* node_set_ids,
    int* num_nodes_per_set,
    int* num_dist_per_set,
    int* node_sets_node_index,
    int* node_sets_dist_index,
    int* node_sets_node_list,
    void* node_sets_dist_fact);              page 91
```

```

int ex_get_concat_side_sets (
    int exoid,
    int* side_set_ids,
    int* num_side_per_set,
    int* num_dist_per_set,
    int* side_sets_elem_index,
    int* side_sets_dist_index,
    int* side_sets_elem_list,
    int* side_sets_side_list,
    void* side_sets_dist_fact);    page 112

int ex_get_coord (
    int exoid,
    void* x_coor,
    void* y_coor,
    void* z_coor);                page 51

int ex_get_coord_names (
    int exoid,
    char** coord_names);         page 55

int ex_get_elem_attr (
    int exoid,
    int elem_blk_id,
    void* attrib);                page 75

int ex_get_elem_blk_ids (
    int exoid,
    int* elem_blk_ids);          page 70

int ex_get_elem_block (
    int exoid,
    int elem_blk_id,
    char* elem_type,
    int* num_elem_this_blk,
    int* num_nodes_per_elem,
    int* num_attr);              page 68

int ex_get_elem_conn (
    int exoid,
    int elem_blk_id,
    int* connect);               page 72

int ex_get_elem_num_map (
    int exoid,
    int* elem_map);              page 61

int ex_get_elem_var (
    int exoid,
    int time_step,
    int elem_var_index,
    int elem_blk_id,
    int num_elem_this_blk,
    void* elem_var_vals);        page 152

```

```

int ex_get_elem_var_tab (
    int exoid,
    int num_elem_blk,
    int num_elem_var,
    int* elem_var_tab);           page 147

int ex_get_elem_var_time (
    int exoid,
    int elem_var_index,
    int elem_number,
    int beg_time_step,
    int end_time_step,
    void* elem_var_vals);       page 154

int ex_get_glob_vars (
    int exoid,
    int time_step,
    int num_glob_vars,
    void* glob_var_vals);       page 159

int ex_get_glob_var_time (
    int exoid,
    int glob_var_index,
    int beg_time_step,
    int end_time_step,
    void* glob_var_vals);       page 161

int ex_get_info (
    int exoid,
    char** info);               page 39

int ex_get_init (
    int exoid,
    char* title,
    int* num_dim,
    int* num_nodes,
    int* num_elem,
    int* num_elem_blk,
    int* num_node_sets,
    int* num_side_sets);        page 31

int ex_get_map (
    int exoid,
    int* elem_map);             page 64

int ex_get_nodal_var (
    int exoid,
    int time_step,
    int nodal_var_index,
    int num_nodes,
    void* nodal_var_vals);      page 165

int ex_get_nodal_var_time (
    int exoid,
    int nodal_var_index,
    int node_number,
    int beg_time_step,
    int end_time_step,
    void* nodal_var_vals);      page 167

```

```

int ex_get_node_num_map (
    int exoid,
    int* node_map);           page 58

int ex_get_node_set (
    int exoid,
    int node_set_id,
    int* node_set_node_list); page 82

int ex_get_node_set_dist_fact (
    int exoid,
    int node_set_id,
    void* node_set_dist_fact); page 85

int ex_get_node_set_ids (
    int exoid,
    int* node_set_ids);      page 86

int ex_get_node_set_param (
    int exoid,
    int node_set_id,
    int* num_nodes_in_set,
    int* num_dist_in_set);   page 79

int ex_get_prop (
    int exoid,
    int obj_type,
    int obj_id,
    char* prop_name,
    int* value);             page 124

int ex_get_prop_array (
    int exoid,
    int obj_type,
    char* prop_name,
    int* values);           page 128

int ex_get_prop_names (
    int exoid,
    int obj_type,
    char** prop_names);     page 120

int ex_get_qa (
    int exoid,
    char* qa_record);       page 35

int ex_get_side_set (
    int exoid,
    int side_set_id,
    int* side_set_elem_list,
    int* side_set_side_list); page 100

int ex_get_side_set_dist_fact (
    int exoid,
    int side_set_id,
    void* side_set_dist_fact); page 104

int ex_get_side_set_ids (
    int exoid
    int* side_set_ids);     page 105

```

```

int ex_get_side_set_node_list (
    int exoid,
    int side_set_id,
    int* side_set_node_cnt_list,
    int* side_set_node_list);    page 106

int ex_get_side_set_param (
    int exoid,
    int side_set_id,
    int* num_side_in_set,
    int* num_dist_fact_in_set);    page 96

int ex_get_time (
    int exoid,
    int time_step,
    void* time_value);    page 141

int ex_get_var_names (
    int exoid,
    char* var_type,
    int num_vars,
    char** var_names);    page 137

int ex_get_var_param (
    int exoid,
    char* var_type,
    int* num_vars);    page 133

int ex_inquire (
    int exoid,
    int req_info,
    int* ret_int,
    float* ret_float,
    char* ret_char);    page 41

int ex_open (
    char* path,
    int mode,
    int* comp_ws,
    int* io_ws,
    float* version);    page 25

int ex_opts (
    int option_val);    page 47

int ex_put_concat_node_sets (
    int exoid,
    int* node_set_ids,
    int* num_nodes_per_set,
    int* num_dist_per_set,
    int* node_sets_node_index,
    int* node_sets_dist_index,
    int* node_sets_node_list,
    void* node_sets_dist_fact);    page 87

```

```

int ex_put_concat_side_sets (
    int exoid,
    int* side_sets_ids,
    int* num_side_per_set,
    int* num_dist_per_set,
    int* side_sets_elem_index,
    int* side_sets_dist_index,
    int* side_sets_elem_list,
    int* side_sets_side_list,
    void* side_sets_dist_fact);    page 108

int ex_put_coord (
    int exoid,
    void* x_coor,
    void* y_coor,
    void* z_coor);                page 49

int ex_put_coord_names (
    int exoid,
    char** coord_names);         page 53

int ex_put_elem_attr (
    int exoid,
    int elem_blk_id,
    void* attrib);               page 73

int ex_put_elem_block (
    int exoid,
    int elem_blk_id,
    char* elem_type,
    int num_elem_this_blk,
    int num_nodes_per_elem,
    int num_attr);               page 65

int ex_put_elem_conn (
    int exoid,
    int elem_blk_id,
    int* connect);               page 71

int ex_put_elem_num_map (
    int exoid,
    int* elem_map);              page 59

int ex_put_elem_var (
    int exoid,
    int time_step,
    int elem_var_index,
    int elem_blk_id,
    int num_elem_this_blk,
    void* elem_var_vals);        page 149

int ex_put_elem_var_tab (
    int exoid,
    int num_elem_blk,
    int num_elem_var,
    int* elem_var_tab);          page 145

int ex_put_glob_vars (
    int exoid,
    int time_step,
    int num_glob_vars,
    void* glob_var_vals);        page 157

```

```

int ex_put_info (
    int exoid,
    int num_info,
    char* info);
    page 37

int ex_put_init (
    int exoid,
    char* title,
    int num_dim,
    int num_nodes,
    int num_elem,
    int num_elem_blk,
    int num_node_sets,
    int num_side_sets);
    page 29

int ex_put_map (
    int exoid,
    int* elem_map);
    page 62

int ex_put_nodal_var (
    int exoid,
    int time_step,
    int nodal_var_index,
    int num_nodes,
    void* nodal_var_vals);
    page 163

int ex_put_node_num_map (
    int exoid,
    int* node_map)
    page 56

int ex_put_node_set (
    int exoid,
    int node_set_id,
    int* node_set_node_list);
    page 81

int ex_put_node_set_dist_fact (
    int exoid,
    int node_set_id,
    void* node_set_dist_fact);
    page 83

int ex_put_node_set_param (
    int exoid,
    int node_set_id,
    int num_nodes_in_set,
    int num_dist_in_set);
    page 77

int ex_put_prop (
    int exoid,
    int obj_type,
    int obj_id,
    char* prop_name,
    int value);
    page 122

int ex_put_prop_array (
    int exoid,
    int obj_type,
    char* prop_name,
    int* values);
    page 126

```

```

int ex_put_prop_names (
    int exoid,
    int obj_type,
    int num_props,
    char** prop_names);           page 118

int ex_put_qa (
    int exoid,
    int num_qa_records,
    char* qa_record);           page 33

int ex_put_side_set (
    int exoid,
    int side_set_id
    int* side_set_elem_list,
    int* side_set_side_list);   page 98

int ex_put_side_set_dist_fact (
    int exoid,
    int side_set_id,
    void* side_set_dist_fact);  page 102

int ex_put_side_set_param (
    int exoid,
    int side_set_id,
    int num_side_in_set,
    int num_dist_fact_in_set);  page 94

int ex_put_var_names (
    int exoid,
    char* var_type,
    int num_vars,
    char** var_names);         page 135

int ex_put_var_param (
    int exoid,
    char* var_type,
    int num_vars);             page 131

int ex_put_time (
    int exoid,
    int time_step,
    void* time_value);         page 139

int ex_update (
    int exoid);                 page 28

```



## FORTRAN binding routines

SUBROUTINE EXCLOS ( IDEXO, IERR) page 27  
    INTEGER IDEXO  
    INTEGER IERR

SUBROUTINE EXCN2S(IDEXO, NSESS, NDESS, IXEESS, IXNESS, LTEESS,  
    LTNESS, LTSESS, IERR) page 115  
    INTEGER IDEXO  
    INTEGER NSESS(\*)  
    INTEGER NDESS(\*)  
    INTEGER IXEESS(\*)  
    INTEGER IXNESS(\*)  
    INTEGER LTEESS(\*)  
    INTEGER LTNESS(\*)  
    INTEGER LTSESS(\*)  
    INTEGER IERR

INTEGER FUNCTION EXCRE (PATH, ICMODE, ICOMPWS, IOWS, IERR) page 23  
    CHARACTER\*(\*) PATH  
    INTEGER ICMODE  
    INTEGER ICOMPWS  
    INTEGER IOWS  
    INTEGER IERR

SUBROUTINE EXERR ( MODNAM, MSG, ERRNUM) page 45  
    CHARACTER\*MXSTLN MODNAM  
    CHARACTER\*MXLNLN MSG  
    INTEGER ERRNUM

SUBROUTINE EXGATM ( IDEXO, TIME, IERR) page 143  
    INTEGER IDEXO  
    REAL TIME(\*)  
    INTEGER IERR

SUBROUTINE EXGCNS ( IDEXO, IDNPSS, NNNPS, NDNPS, IXNNPS, IXDNPS,  
    LTNNPS, FACNPS, IERR) page 91  
    INTEGER IDEXO  
    INTEGER IDNPSS(\*)  
    INTEGER NNNPS(\*)  
    INTEGER NDNPS(\*)  
    INTEGER IXNNPS(\*)  
    INTEGER IXDNPS(\*)  
    INTEGER LTNNPS(\*)  
    REAL FACNPS(\*)  
    INTEGER IERR

SUBROUTINE EXGCON ( IDEXO, NAMECO, IERR) page 55  
    INTEGER IDEXO  
    CHARACTER\*MXSTLN NAMECO(\*)  
    INTEGER IERR

SUBROUTINE EXGCOR ( IDEXO, XN, YN, ZN, IERR) page 51  
    INTEGER IDEXO  
    REAL XN(\*)  
    REAL YN(\*)  
    REAL ZN(\*)  
    INTEGER IERR

SUBROUTINE EXGCSS ( IDEXO, IDESSS, NSESS, NDESS, IXEESS, IXDESS,  
LTHESS, LTSESS, FACESS, IERR) page 112

```

    INTEGER IDEXO
    INTEGER IDESSS(*)
    INTEGER NSESS(*)
    INTEGER NDESS(*)
    INTEGER IXEESS(*)
    INTEGER IXDESS(*)
    INTEGER LTHESS(*)
    INTEGER LTSESS(*)
    REAL FACESS(*)
    INTEGER IERR

```

SUBROUTINE EXGEAT ( IDEXO, IDELB, ATRIB, IERR) page 75

```

    INTEGER IDEXO
    INTEGER IDELB
    REAL ATRIB(*)
    INTEGER IERR

```

SUBROUTINE EXGEBI ( IDEXO, IDELBS, IERR) page 70

```

    INTEGER IDEXO
    INTEGER IDELBS(*)
    INTEGER IERR

```

SUBROUTINE EXGELB ( IDEXO, IDELB, NAMELB, NUMELB, NUMLNK, NUMATR,  
IERR) page 68

```

    INTEGER IDEXO
    INTEGER IDELB
    CHARACTER*MXSTLN NAMELB
    INTEGER NUMELB
    INTEGER NUMLNK
    INTEGER NUMATR
    INTEGER IERR

```

SUBROUTINE EXGELC ( IDEXO, IDELB, LINK, IERR) page 72

```

    INTEGER IDEXO
    INTEGER IDELB
    INTEGER LINK(*)
    INTEGER IERR

```

SUBROUTINE EXGENM ( IDEXO, MAPEL, IERR) page 61

```

    INTEGER IDEXO
    INTEGER MAPEL(*)
    INTEGER IERR

```

SUBROUTINE EXGEV ( IDEXO, ISTEP, IXELEV, IDELB, NUMELB, VALEV, IERR)  
page 152

```

    INTEGER IDEXO
    INTEGER ISTEP
    INTEGER IXELEV
    INTEGER IDELB
    INTEGER NUMELB
    REAL VALEV(*)
    INTEGER IERR

```

SUBROUTINE EXGEVT ( IDEXO, IXELEV, IELNUM, ISTEPB, ISTPE, VALEV, IERR) page 154

INTEGER IDEXO  
INTEGER IXELEV  
INTEGER IELNUM  
INTEGER ISTEPB  
INTEGER ISTPE  
REAL VALEV(\*)  
INTEGER IERR

SUBROUTINE EXGGV ( IDEXO, ISTEP, NVARGL, VALGV, IERR) page 159

INTEGER IDEXO  
INTEGER ISTEP  
INTEGER NVARGL  
REAL VALGV(\*)  
INTEGER IERR

SUBROUTINE EXGGVT ( IDEXO, IXGLOV, ISTEPB, ISTPE, VALGV, IERR) page 161

INTEGER IDEXO  
INTEGER IXGLOV  
INTEGER ISTEPB  
INTEGER ISTPE  
REAL VALGV(\*)  
INTEGER IERR

SUBROUTINE EXGINF ( IDEXO, INFO, IERR) page 39

INTEGER IDEXO  
CHARACTER\*MXLNLN INFO(\*)  
INTEGER IERR

SUBROUTINE EXGINI ( IDEXO, TITLE, NDIM, NUMNP, NUMEL, NELBLK, NUMNPS, NUMESS, IERR) page 31

INTEGER IDEXO  
CHARACTER\*MXLNLN TITLE  
INTEGER NDIM  
INTEGER NUMNP  
INTEGER NUMEL  
INTEGER NELBLK  
INTEGER NUMNPS  
INTEGER NUMESS  
INTEGER IERR

SUBROUTINE EXGMAP ( IDEXO, MAPEL, IERR) page 64

INTEGER IDEXO  
INTEGER MAPEL(\*)  
INTEGER IERR

SUBROUTINE EXGNNM ( IDEXO, MAPNOD, IERR) page 58

INTEGER IDEXO  
INTEGER MAPNOD(\*)  
INTEGER IERR

SUBROUTINE EXGNP ( IDEXO, IDNPS, NNNPS, NDNPS, IERR) page 79

INTEGER IDEXO  
INTEGER IDNPS  
INTEGER NNNPS  
INTEGER NDNPS  
INTEGER IERR

SUBROUTINE EXGNS ( IDEXO, IDNPS, LTNNPS, IERR) page 82  
 INTEGER IDEXO  
 INTEGER IDNPS  
 INTEGER LTNNPS(\*)  
 INTEGER IERR

SUBROUTINE EXGNSD ( IDEXO, IDNPS, FACNPS, IERR) page 85  
 INTEGER IDEXO  
 INTEGER IDNPS  
 REAL FACNPS(\*)  
 INTEGER IERR

SUBROUTINE EXGNSI ( IDEXO, IDNPSS, IERR) page 86  
 INTEGER IDEXO  
 INTEGER IDNPSS(\*)  
 INTEGER IERR

SUBROUTINE EXGNV ( IDEXO, ISTEP, IXNODV, NUMNP, VALNV, IERR) page 165  
 INTEGER IDEXO  
 INTEGER ISTEP  
 INTEGER IXNODV  
 INTEGER NUMNP  
 REAL VALNV(\*)  
 INTEGER IERR

SUBROUTINE EXGNVT ( IDEXO, IXNODV, NODNUM, ISTEP, ISTPE, VALNV, IERR) page 167  
 INTEGER IDEXO  
 INTEGER IXNODV  
 INTEGER NODNUM  
 INTEGER ISTEP  
 INTEGER ISTPE  
 REAL VALNV(\*)  
 INTEGER IERR

SUBROUTINE EXGP ( IDEXO, ITYPE, ID, NAMEPR, IVAL, IERR) page 124  
 INTEGER IDEXO  
 INTEGER ITYPE  
 INTEGER ID  
 CHARACTER\*MXSTLN NAMEPR  
 INTEGER IVAL  
 INTEGER IERR

SUBROUTINE EXGPA ( IDEXO, ITYPE, NAMEPR, IVALS, IERR) page 128  
 INTEGER IDEXO  
 INTEGER ITYPE  
 CHARACTER\*MXSTLN NAMEPR  
 INTEGER IVAL(\*)  
 INTEGER IERR

SUBROUTINE EXGPN ( IDEXO, ITYPE, NAMEPR, IERR) page 120  
 INTEGER IDEXO  
 INTEGER ITYPE  
 CHARACTER\*MXSTLN NAMEPR(\*)  
 INTEGER IERR

SUBROUTINE EXGQA ( IDEXO, QAREC, IERR) page 35  
 INTEGER IDEXO  
 CHARACTER\*MXSTLN QAREC(4,\*)  
 INTEGER IERR

SUBROUTINE EXGSP ( IDEXO, IDESS, NSESS, NDESS, IERR) page 96  
 INTEGER IDEXO  
 INTEGER IDESS  
 INTEGER NSESS  
 INTEGER NDESS  
 INTEGER IERR

SUBROUTINE EXGSS ( IDEXO, IDESS, LTEESS, LTSESS, IERR) page 100  
 INTEGER IDEXO  
 INTEGER IDESS  
 INTEGER LTEESS(\*)  
 INTEGER LTSESS(\*)  
 INTEGER IERR

SUBROUTINE EXGSSD ( IDEXO, IDESS, FACESS, IERR) page 104  
 INTEGER IDEXO  
 INTEGER IDESS  
 REAL FACESS(\*)  
 INTEGER IERR

SUBROUTINE EXGSSI ( IDEXO, IDESSS, IERR) page 105  
 INTEGER IDEXO  
 INTEGER IDESSS(\*)  
 INTEGER IERR

SUBROUTINE EXGSSN ( IDEXO, IDESS, INCNT, LTNESS, IERR) page 106  
 INTEGER IDEXO  
 INTEGER IDESS  
 INCNT(\*)  
 INTEGER LTNESS(\*)  
 INTEGER IERR

SUBROUTINE EXGTIM ( IDEXO, NSTEP, TIME, IERR) page 141  
 INTEGER IDEXO  
 INTEGER NSTEP  
 REAL TIME  
 INTEGER IERR

SUBROUTINE EXGVAN ( IDEXO, VARTYP, NVAR, NAMES, IERR) page 137  
 INTEGER IDEXO  
 CHARACTER\*1 VARTYP  
 INTEGER NVAR  
 CHARACTER\*MXSTLN NAMES(\*)  
 INTEGER IERR

SUBROUTINE EXGVP ( IDEXO, VARTYP, NVAR, IERR) page 133  
 INTEGER IDEXO  
 CHARACTER\*1 VARTYP  
 INTEGER NVAR  
 INTEGER IERR

SUBROUTINE EXGVTT ( IDEXO, NELBLK, NVAREL, ISEVOK, IERR) page 147  
 INTEGER IDEXO  
 INTEGER NELBLK  
 INTEGER NVAREL  
 INTEGER ISEVOK(NVAREL, NELBLK)  
 INTEGER IERR

SUBROUTINE EXINQ ( IDEXO, INFREQ, INTRET, RELRET, CHRRET, IERR) page 41

INTEGER IDEXO  
INTEGER INFREQ  
INTEGER INTRET  
REAL RELRET  
CHARACTER\*(\*) CHRRET  
INTEGER IERR

INTEGER FUNCTION EXOPEN (PATH, IMODE, ICOMPWS, IOWS, VERS, IERR) page 25

CHARACTER\*(\*) PATH  
INTEGER IMODE  
INTEGER ICOMPWS  
INTEGER IOWS  
REAL VERS  
INTEGER IERR

SUBROUTINE EXOPTS ( OPTVAL, IERR) page 47

INTEGER OPTVAL  
INTEGER IERR

SUBROUTINE EXPCNS ( IDEXO, IDNPSS, NNNPS, NDNPS, IXNNPS, IXDNPS, LTNNPS, FACNPS, IERR) page 87

INTEGER IDEXO  
INTEGER IDNPSS(\*)  
INTEGER NNNPS(\*)  
INTEGER NDNPS(\*)  
INTEGER IXNNPS(\*)  
INTEGER IXDNPS(\*)  
INTEGER LTNNPS(\*)  
REAL FACNPS(\*)  
INTEGER IERR

SUBROUTINE EXPCON ( IDEXO, NAMECO, IERR) page 53

INTEGER IDEXO  
CHARACTER\*MXSTLN NAMECO(\*)  
INTEGER IERR

SUBROUTINE EXPCOR ( IDEXO, XN, YN, ZN, IERR) page 49

INTEGER IDEXO  
REAL XN(\*)  
REAL YN(\*)  
REAL ZN(\*)  
INTEGER IERR

SUBROUTINE EXPCSS ( IDEXO, IDESSS, NSESS, NDESS, IXEESS, IXDESS, LTEESS, LTSESS, FACESS, IERR) page 108

INTEGER IDEXO  
INTEGER IDESSS(\*)  
INTEGER NSESS(\*)  
INTEGER NDESS(\*)  
INTEGER IXEESS(\*)  
INTEGER IXDESS(\*)  
INTEGER LTEESS(\*)  
INTEGER LTSESS(\*)  
REAL FACESS(\*)  
INTEGER IERR

SUBROUTINE EXPEAT ( IDEXO, IDELB, ATRIB, IERR) page 73  
INTEGER IDEXO  
INTEGER IDELB  
REAL ATRIB(\*)  
INTEGER IERR

SUBROUTINE EXPELB ( IDEXO, IDELB, NAMELB, NUMELB, NUMLNK, NUMATR, IERR) page 65  
INTEGER IDEXO  
INTEGER IDELB  
CHARACTER\*MXSTLN NAMELB  
INTEGER NUMELB  
INTEGER NUMLNK  
INTEGER NUMATR  
INTEGER IERR

SUBROUTINE EXPENM ( IDEXO, MAPEL, IERR) page 59  
INTEGER IDEXO  
INTEGER MAPEL(\*)  
INTEGER IERR

SUBROUTINE EXPELC ( IDEXO, IDELB, LINK, IERR) page 71  
INTEGER IDEXO  
INTEGER IDELB  
INTEGER LINK(\*)  
INTEGER IERR

SUBROUTINE EXPEV ( IDEXO, ISTEP, IXELEV, IDELB, NUMELB, VALEV, IERR) page 149  
INTEGER IDEXO  
INTEGER ISTEP  
INTEGER IXELEV  
INTEGER IDELB  
INTEGER NUMELB  
REAL VALEV(\*)  
INTEGER IERR

SUBROUTINE EXPGV ( IDEXO, ISTEP, NVARGL, VALGV, IERR) page 157  
INTEGER IDEXO  
INTEGER ISTEP  
INTEGER NVARGL  
REAL VALGV(\*)  
INTEGER IERR

SUBROUTINE EXPINF ( IDEXO, NINFO, INFO, IERR) page 37  
INTEGER IDEXO  
INTEGER NINFO  
CHARACTER\*MXLNLN INFO(\*)  
INTEGER IERR

SUBROUTINE EXPINI ( IDEXO, TITLE, NDIM, NUMNP, NUMEL, NELBLK, NUMNPS, NUMESS, IERR) page 29  
INTEGER IDEXO  
CHARACTER\*MXLNLN TITLE  
INTEGER NDIM  
INTEGER NUMNP  
INTEGER NUMEL  
INTEGER NELBLK  
INTEGER NUMNPS  
INTEGER NUMESS  
INTEGER IERR

SUBROUTINE EXPMAP ( IDEXO, MAPEL, IERR) page 62  
 INTEGER IDEXO  
 INTEGER MAPEL(\*)  
 INTEGER IERR

SUBROUTINE EXPNNM ( IDEXO, MAPNOD, IERR) page 56  
 INTEGER IDEXO  
 INTEGER MAPNOD(\*)  
 INTEGER IERR

SUBROUTINE EXPNP ( IDEXO, IDNPS, NNNPS, NDNPS, IERR) page 77  
 INTEGER IDEXO  
 INTEGER IDNPS  
 INTEGER NNNPS  
 INTEGER NDNPS  
 INTEGER IERR

SUBROUTINE EXPNS ( IDEXO, IDNPS, LTNNPS, IERR) page 81  
 INTEGER IDEXO  
 INTEGER IDNPS  
 INTEGER LTNNPS(\*)  
 INTEGER IERR

SUBROUTINE EXPNSD ( IDEXO, IDNPS, FACNPS, IERR) page 83  
 INTEGER IDEXO  
 INTEGER IDNPS  
 REAL FACNPS(\*)  
 INTEGER IERR

SUBROUTINE EXPNV ( IDEXO, ISTEP, IXNODV, NUMNP, VALNV, IERR) page 163  
 INTEGER IDEXO  
 INTEGER ISTEP  
 INTEGER IXNODV  
 INTEGER NUMNP  
 REAL VALNV(\*)  
 INTEGER IERR

SUBROUTINE EXPP ( IDEXO, ITYPE, ID, NAMEPR, IVAL, IERR) page 122  
 INTEGER IDEXO  
 INTEGER ITYPE  
 INTEGER ID  
 CHARACTER\*MXSTLN NAMEPR  
 INTEGER IVAL  
 INTEGER IERR

SUBROUTINE EXPPA ( IDEXO, ITYPE, NAMEPR, IVALS, IERR) page 126  
 INTEGER IDEXO  
 INTEGER ITYPE  
 CHARACTER\*MXSTLN NAMEPR  
 INTEGER IVAL(\*)  
 INTEGER IERR



SUBROUTINE EXPPN ( IDEXO, ITYPE, NPROPS, NAMEPR, IERR) page 118  
 INTEGER IDEXO  
 INTEGER ITYPE  
 INTEGER NPROPS  
 CHARACTER\*MXSTLN NAMEPR(\*)  
 INTEGER IERR

SUBROUTINE EXPQA ( IDEXO, NQAREC, QAREC, IERR) page 33  
 INTEGER IDEXO  
 INTEGER NQAREC  
 CHARACTER\*MXSTLN QAREC (4,\*)  
 INTEGER IERR

SUBROUTINE EXPSP ( IDEXO, IDESS, NSESS, NDESS, IERR) page 94  
 INTEGER IDEXO  
 INTEGER IDESS  
 INTEGER NSESS  
 INTEGER NDESS  
 INTEGER IERR

SUBROUTINE EXPSS ( IDEXO, IDESS, LTEESS, LTSESS, IERR) page 98  
 INTEGER IDEXO  
 INTEGER IDESS  
 INTEGER LTEESS(\*)  
 INTEGER LTSESS(\*)  
 INTEGER IERR

SUBROUTINE EXPSSD ( IDEXO, IDESS, FACESS, IERR) page 102  
 INTEGER IDEXO  
 INTEGER IDESS  
 REAL FACESS(\*)  
 INTEGER IERR

SUBROUTINE EXPTIM ( IDEXO, NSTEP, TIME, IERR) page 139  
 INTEGER IDEXO  
 INTEGER NSTEP  
 REAL TIME  
 INTEGER IERR

SUBROUTINE EXPVAN ( IDEXO, VARTYP, NVAR, NAMES, IERR) page 135  
 INTEGER IDEXO  
 CHARACTER\*1 VARTYP  
 INTEGER NVAR  
 CHARACTER\*MXSTLN NAMES(\*)  
 INTEGER IERR

SUBROUTINE EXPVP ( IDEXO, VARTYP, NVAR, IERR) page 131  
 INTEGER IDEXO  
 CHARACTER\*1 VARTYP  
 INTEGER NVAR  
 INTEGER IERR

SUBROUTINE EXPVTT ( IDEXO, NELBLK, NVAREL, ISEVOK, IERR) page 145  
 INTEGER IDEXO  
 INTEGER NELBLK  
 INTEGER NVAREL  
 INTEGER ISEVOK(NVAREL,NELBLK)  
 INTEGER IERR

SUBROUTINE EXUPDA ( IDEXO, IERR) page 28  
 INTEGER IDEXO  
 INTEGER IERR

Intentionally Left Blank

# Appendix C

## Error Messages

This appendix contains descriptions of error codes that are returned by the EXODUS II library routines.

The following are return codes that are specific to EXODUS II routines. The error names are defined constants (in `exodusII.h` for C and `exodusII.inc` for Fortran) currently assigned the specified values. A 0 (zero) means no error; a positive number is a warning; a negative number is a fatal error.

Error Name (C)	Error Name (Fortran)	Value	Description
EX_FATAL	EXFATL	-1	fatal error flag
EX_OK	EXOK	0	no error flag
EX_WARN	EXWARN	1	warning flag
EX_MEMFAIL	EXMEMF	-100	memory allocation failure flag
EX_BADFILEMODE	EXBFMD	-101	bad file mode
EX_BADFILEID	EXBFID	-102	bad file id; usually an unopened file
EX_WRONGFILETYPE		-103	wrong file type for function
EX_LOOKUPFAIL	EXBTID	-104	property table lookup failed
EX_BADPARAM	EXBPRM	-105	bad parameter passed
EX_MSG	EXPMSG	100	user-defined message
EX_PRTLASTMSG	EXLMSG	101	print last error message msg code

The following are codes returned by netCDF functions. The error names are defined constants (in `netcdf.h`) currently set to the specified values.

Error Name	Value	Description
NC_NOERR	0	No error
NC_EBADID	1	Not a netcdf id
NC_ENFILE	2	Too many netcdfs open
NC_EEXIST	3	netcdf file exists && NC_NOCLOBBER
NC_EINVAL	4	Invalid argument
NC_EPERM	5	Write to read only file
NC_ENOTINDEFINE	6	Operation not allowed in data mode
NC_EINDEFINE	7	Operation not allowed in define mode
NC_EINVALCOORDS	8	Coordinates out of domain
NC_EMAXDIMS	9	MAX_NC_DIMS (defined in netcdf.h) exceeded
NC_ENAMEINUSE	10	String match to name in use
NC_ENOTATT	11	Attribute not found
NC_EMAXATTRS	12	MAX_NC_ATTRS (defined in netcdf.h) exceeded
NC_EBADTYPE	13	Not a netcdf data type
NC_EBADDIM	14	Invalid dimension id
NC_EUNLIMPOS	15	NC_UNLIMITED in the wrong index
NC_EMAXVARS	16	MAX_NC_VARS (defined in netcdf.h) exceeded
NC_ENOTVAR	17	Variable not found
NC_EGLOBAL	18	Action prohibited on NC_GLOBAL varid
NC_ENOTNC	19	Not a netcdf file
NC_ESTS	20	In Fortran, string too short
NC_EMAXNAME	21	MAX_NC_NAME (defined in netcdf.h) exceeded
NC_EUNLIMIT	22	NC_UNLIMITED size already in use
NC_EXDR	32	XDR error
NC_SYSERR	-1	Fatal system error

# Appendix D

## Sample Codes

This appendix contains examples of C and Fortran programs that use the EXODUS II API.

### C Write Example Code

The following is a C program that creates and populates an EXODUS II file:

```
#include <stdio.h>
#include "netcdf.h"
#include "exodusII.h"

main ()
{
    int exoid, num_dim, num_nodes, num_elem, num_elem_blk;
    int num_elem_in_block[10], num_nodes_per_elem[10];
    int num_node_sets, num_sides, num_side_sets, error;
    int i, j, k, m, *elem_map, *connect;
    int node_list[100],elem_list[100],side_list[100];
    int ebids[10], ids[10];
    int num_sides_per_set[10], num_nodes_per_set[10], num_elem_per_set[10];
    int num_df_per_set[10];
    int df_ind[10], node_ind[10], elem_ind[10], side_ind[10];
    int num_qa_rec, num_info;
    int num_glo_vars, num_nod_vars, num_ele_vars;
    int *truth_tab;
    int whole_time_step, num_time_steps;
    int ndims, nvars, ngatts, recdim;
    int CPU_word_size,IO_word_size;
    int prop_array[2];

    float *glob_var_vals, *nodal_var_vals, *elem_var_vals;
    float time_value;
    float x[100], y[100], z[100], *dummy;
    float attrib[1], dist_fact[100];
    char *coord_names[3], *qa_record[2][4], *info[3], *var_names[3];
    char tmpstr[80];
    char *prop_names[2];

    dummy = 0; /* assign this so the Cray compiler doesn't complain */

    /* Specify compute and i/o word size */

    CPU_word_size = 0; /* float or double */
    IO_word_size = 0; /* use system default (4 bytes) */

    /* create EXODUS II file */

    exoid = ex_create ("test.exo", /* filename path */
                      EX_CLOBBER, /* create mode */
                      &CPU_word_size, /* CPU float word size in bytes */
                      &IO_word_size); /* I/O float word size in bytes */
    /* ncopts = NC_VERBOSE; */

    /* initialize file with parameters */

    num_dim = 3;
    num_nodes = 26;
```

```

num_elem = 5;
num_elem_blk = 5;
num_node_sets = 2;
num_side_sets = 5;

error = ex_put_init (exoid, "This is a test", num_dim, num_nodes, num_elem,
                    num_elem_blk, num_node_sets, num_side_sets);

/* write nodal coordinates values and names to database */

/* Quad #1 */
x[0] = 0.0; y[0] = 0.0; z[0] = 0.0;
x[1] = 1.0; y[1] = 0.0; z[1] = 0.0;
x[2] = 1.0; y[2] = 1.0; z[2] = 0.0;
x[3] = 0.0; y[3] = 1.0; z[3] = 0.0;

/* Quad #2 */
x[4] = 1.0; y[4] = 0.0; z[4] = 0.0;
x[5] = 2.0; y[5] = 0.0; z[5] = 0.0;
x[6] = 2.0; y[6] = 1.0; z[6] = 0.0;
x[7] = 1.0; y[7] = 1.0; z[7] = 0.0;

/* Hex #1 */
x[8] = 0.0; y[8] = 0.0; z[8] = 0.0;
x[9] = 10.0; y[9] = 0.0; z[9] = 0.0;
x[10] = 10.0; y[10] = 0.0; z[10] = -10.0;
x[11] = 1.0; y[11] = 0.0; z[11] = -10.0;
x[12] = 1.0; y[12] = 10.0; z[12] = 0.0;
x[13] = 10.0; y[13] = 10.0; z[13] = 0.0;
x[14] = 10.0; y[14] = 10.0; z[14] = -10.0;
x[15] = 1.0; y[15] = 10.0; z[15] = -10.0;

/* Tetra #1 */
x[16] = 0.0; y[16] = 0.0; z[16] = 0.0;
x[17] = 1.0; y[17] = 0.0; z[17] = 5.0;
x[18] = 10.0; y[18] = 0.0; z[18] = 2.0;
x[19] = 7.0; y[19] = 5.0; z[19] = 3.0;

/* Wedge #1 */
x[20] = 3.0; y[20] = 0.0; z[20] = 6.0;
x[21] = 6.0; y[21] = 0.0; z[21] = 0.0;
x[22] = 0.0; y[22] = 0.0; z[22] = 0.0;
x[23] = 3.0; y[23] = 2.0; z[23] = 6.0;
x[24] = 6.0; y[24] = 2.0; z[24] = 2.0;
x[25] = 0.0; y[25] = 2.0; z[25] = 0.0;

error = ex_put_coord (exoid, x, y, z);

coord_names[0] = "xcoor";
coord_names[1] = "ycoor";
coord_names[2] = "zcoor";

error = ex_put_coord_names (exoid, coord_names);

/* write element order map */

elem_map = (int *) calloc(num_elem, sizeof(int));

for (i=1; i<=num_elem; i++)
{
    elem_map[i-1] = i;
}

error = ex_put_map (exoid, elem_map);

free (elem_map);

/* write element block parameters */

num_elem_in_block[0] = 1;

```

```

num_elem_in_block[1] = 1;
num_elem_in_block[2] = 1;
num_elem_in_block[3] = 1;
num_elem_in_block[4] = 1;

num_nodes_per_elem[0] = 4; /* elements in block #1 are 4-node quads */
num_nodes_per_elem[1] = 4; /* elements in block #2 are 4-node quads */
num_nodes_per_elem[2] = 8; /* elements in block #3 are 8-node hexes */
num_nodes_per_elem[3] = 4; /* elements in block #3 are 4-node tetras */
num_nodes_per_elem[4] = 6; /* elements in block #3 are 6-node wedges */

ebids[0] = 10;
ebids[1] = 11;
ebids[2] = 12;
ebids[3] = 13;
ebids[4] = 14;

error = ex_put_elem_block (exoid, ebids[0], "QUAD", num_elem_in_block[0],
                           num_nodes_per_elem[0], 1);

error = ex_put_elem_block (exoid, ebids[1], "QUAD", num_elem_in_block[1],
                           num_nodes_per_elem[1], 1);

error = ex_put_elem_block (exoid, ebids[2], "HEX", num_elem_in_block[2],
                           num_nodes_per_elem[2], 1);

error = ex_put_elem_block (exoid, ebids[3], "TETRA", num_elem_in_block[3],
                           num_nodes_per_elem[3], 1);

error = ex_put_elem_block (exoid, ebids[4], "WEDGE", num_elem_in_block[4],
                           num_nodes_per_elem[4], 1);

/* write element block properties */

prop_names[0] = "TOP";
prop_names[1] = "RIGHT";
error = ex_put_prop_names(exoid, EX_ELEM_BLOCK, 2, prop_names);

error = ex_put_prop(exoid, EX_ELEM_BLOCK, ebids[0], "TOP", 1);
error = ex_put_prop(exoid, EX_ELEM_BLOCK, ebids[1], "TOP", 1);
error = ex_put_prop(exoid, EX_ELEM_BLOCK, ebids[2], "RIGHT", 1);
error = ex_put_prop(exoid, EX_ELEM_BLOCK, ebids[3], "RIGHT", 1);
error = ex_put_prop(exoid, EX_ELEM_BLOCK, ebids[4], "RIGHT", 1);

/* write element connectivity */

connect = (int *) calloc(8, sizeof(int));
connect[0] = 1; connect[1] = 2; connect[2] = 3; connect[3] = 4;

error = ex_put_elem_conn (exoid, ebids[0], connect);

connect[0] = 5; connect[1] = 6; connect[2] = 7; connect[3] = 8;

error = ex_put_elem_conn (exoid, ebids[1], connect);

connect[0] = 9; connect[1] = 10; connect[2] = 11; connect[3] = 12;
connect[4] = 13; connect[5] = 14; connect[6] = 15; connect[7] = 16;

error = ex_put_elem_conn (exoid, ebids[2], connect);

connect[0] = 17; connect[1] = 18; connect[2] = 19; connect[3] = 20;

error = ex_put_elem_conn (exoid, ebids[3], connect);

connect[0] = 21; connect[1] = 22; connect[2] = 23;
connect[3] = 24; connect[4] = 25; connect[5] = 26;

error = ex_put_elem_conn (exoid, ebids[4], connect);

free (connect);

/* write element block attributes */

```

```

attrib[0] = 3.14159;
error = ex_put_elem_attr (exoid, ebids[0], attrib);

attrib[0] = 6.14159;
error = ex_put_elem_attr (exoid, ebids[1], attrib);

error = ex_put_elem_attr (exoid, ebids[2], attrib);
error = ex_put_elem_attr (exoid, ebids[3], attrib);
error = ex_put_elem_attr (exoid, ebids[4], attrib);
/* write individual node sets */

error = ex_put_node_set_param (exoid, 20, 5, 5);

node_list[0] = 100; node_list[1] = 101; node_list[2] = 102;
node_list[3] = 103; node_list[4] = 104;

dist_fact[0] = 1.0; dist_fact[1] = 2.0; dist_fact[2] = 3.0;
dist_fact[3] = 4.0; dist_fact[4] = 5.0;

error = ex_put_node_set (exoid, 20, node_list);
error = ex_put_node_set_dist_fact (exoid, 20, dist_fact);

error = ex_put_node_set_param (exoid, 21, 3, 3);

node_list[0] = 200; node_list[1] = 201; node_list[2] = 202;

dist_fact[0] = 1.1; dist_fact[1] = 2.1; dist_fact[2] = 3.1;

error = ex_put_node_set (exoid, 21, node_list);
error = ex_put_node_set_dist_fact (exoid, 21, dist_fact);

error = ex_put_prop(exoid, EX_NODE_SET, 20, "FACE", 4);
error = ex_put_prop(exoid, EX_NODE_SET, 21, "FACE", 5);

prop_array[0] = 1000;
prop_array[1] = 2000;

error = ex_put_prop_array(exoid, EX_NODE_SET, "VELOCITY", prop_array);
/* write concatenated node sets; this produces the same information as
 * the above code which writes individual node sets
 */
/* THIS SECTION IS COMMENTED OUT

ids[0] = 20; ids[1] = 21;

num_nodes_per_set[0] = 5; num_nodes_per_set[1] = 3;

node_ind[0] = 0; node_ind[1] = 5;

node_list[0] = 100; node_list[1] = 101; node_list[2] = 102;
node_list[3] = 103; node_list[4] = 104;
node_list[5] = 200; node_list[6] = 201; node_list[7] = 202;

num_df_per_set[0] = 5; num_df_per_set[1] = 3;

df_ind[0] = 0; df_ind[1] = 5;

dist_fact[0] = 1.0; dist_fact[1] = 2.0; dist_fact[2] = 3.0;
dist_fact[3] = 4.0; dist_fact[4] = 5.0;
dist_fact[5] = 1.1; dist_fact[6] = 2.1; dist_fact[7] = 3.1;

error = ex_put_concat_node_sets (exoid, ids, num_nodes_per_set,
                                num_df_per_set, node_ind,
                                df_ind, node_list, dist_fact);

error = ex_put_prop(exoid, EX_NODE_SET, 20, "FACE", 4);
error = ex_put_prop(exoid, EX_NODE_SET, 21, "FACE", 5);

```



```

prop_array[0] = 1000;
prop_array[1] = 2000;

error = ex_put_prop_array(exoid, EX_NODE_SET, "VELOCITY", prop_array);

END COMMENTED OUT SECTION */

/* write individual side sets */

/* side set #1 - quad */
error = ex_put_side_set_param (exoid, 30, 2, 4);

elem_list[0] = 2; elem_list[1] = 2;

side_list[0] = 4; side_list[1] = 2;

dist_fact[0] = 30.0; dist_fact[1] = 30.1; dist_fact[2] = 30.2;
dist_fact[3] = 30.3;

error = ex_put_side_set (exoid, 30, elem_list, side_list);
error = ex_put_side_set_dist_fact (exoid, 30, dist_fact);

/* side set #2 - quad, spanning 2 elements */
error = ex_put_side_set_param (exoid, 31, 2, 4);

elem_list[0] = 1; elem_list[1] = 2;

side_list[0] = 2; side_list[1] = 3;

dist_fact[0] = 31.0; dist_fact[1] = 31.1; dist_fact[2] = 31.2;
dist_fact[3] = 31.3;

error = ex_put_side_set (exoid, 31, elem_list, side_list);
error = ex_put_side_set_dist_fact (exoid, 31, dist_fact);

/* side set #3 - hex */
error = ex_put_side_set_param (exoid, 32, 7, 0);

elem_list[0] = 3; elem_list[1] = 3;
elem_list[2] = 3; elem_list[3] = 3;
elem_list[4] = 3; elem_list[5] = 3;
elem_list[6] = 3;

side_list[0] = 5; side_list[1] = 3;
side_list[2] = 3; side_list[3] = 2;
side_list[4] = 4; side_list[5] = 1;
side_list[6] = 6;

error = ex_put_side_set (exoid, 32, elem_list, side_list);

/* side set #4 - tetras */
error = ex_put_side_set_param (exoid, 33, 4, 0);

elem_list[0] = 4; elem_list[1] = 4;
elem_list[2] = 4; elem_list[3] = 4;

side_list[0] = 1; side_list[1] = 2;
side_list[2] = 3; side_list[3] = 4;

error = ex_put_side_set (exoid, 33, elem_list, side_list);

/* side set #5 - wedges */
error = ex_put_side_set_param (exoid, 34, 5, 0);

```

```

elem_list[0] = 5; elem_list[1] = 5;
elem_list[2] = 5; elem_list[3] = 5;
elem_list[4] = 5;

side_list[0] = 1; side_list[1] = 2;
side_list[2] = 3; side_list[3] = 4;
side_list[4] = 5;

error = ex_put_side_set (exoid, 34, elem_list, side_list);

/* write concatenated side sets; side set node lists (which is how side sets
 * were described in EXODUS I) are converted to side set side lists and then
 * written out; this produces the same information as the above code which
 * writes individual side sets
 */

/* THIS SECTION IS COMMENTED OUT

ids[0] = 30;
ids[1] = 31;
ids[2] = 32;
ids[3] = 33;
ids[4] = 34;

node_list[0] = 8; node_list[1] = 5;
node_list[2] = 6; node_list[3] = 7;

node_list[4] = 2; node_list[5] = 3;
node_list[6] = 7; node_list[7] = 8;

node_list[8] = 9; node_list[9] = 12;
node_list[10] = 11; node_list[11] = 10;

node_list[12] = 11; node_list[13] = 12;
node_list[14] = 16; node_list[15] = 15;

node_list[16] = 16; node_list[17] = 15;
node_list[18] = 11; node_list[19] = 12;

node_list[20] = 10; node_list[21] = 11;
node_list[22] = 15; node_list[23] = 14;

node_list[24] = 13; node_list[25] = 16;
node_list[26] = 12; node_list[27] = 9;

node_list[28] = 14; node_list[29] = 13;
node_list[30] = 9; node_list[31] = 10;

node_list[32] = 16; node_list[33] = 13;
node_list[34] = 14; node_list[35] = 15;

node_list[36] = 17; node_list[37] = 18;
node_list[38] = 20;

node_list[39] = 18; node_list[40] = 19;
node_list[41] = 20;

node_list[42] = 20; node_list[43] = 19;
node_list[44] = 17;

node_list[45] = 19; node_list[46] = 18;
node_list[47] = 17;

node_list[48] = 25; node_list[49] = 24;
node_list[50] = 21; node_list[51] = 22;

node_list[52] = 26; node_list[53] = 25;
node_list[54] = 22; node_list[55] = 23;

node_list[56] = 26; node_list[57] = 23;
node_list[58] = 21; node_list[59] = 24;

```

```

node_list[60] = 23; node_list[61] = 22;
node_list[62] = 21;

node_list[63] = 24; node_list[64] = 25;
node_list[65] = 26;

node_ind[0] = 0;
node_ind[1] = 4;
node_ind[2] = 8;
node_ind[3] = 36;
node_ind[4] = 47;

num_elem_per_set[0] = 2;
num_elem_per_set[1] = 2;
num_elem_per_set[2] = 7;
num_elem_per_set[3] = 4;
num_elem_per_set[4] = 5;

num_nodes_per_set[0] = 4;
num_nodes_per_set[1] = 4;
num_nodes_per_set[2] = 28;
num_nodes_per_set[3] = 12;
num_nodes_per_set[4] = 18;

elem_ind[0] = 0;
elem_ind[1] = 2;
elem_ind[2] = 4;
elem_ind[3] = 11;
elem_ind[4] = 15;

elem_list[0] = 2; elem_list[1] = 2;
elem_list[2] = 1; elem_list[3] = 2;
elem_list[4] = 3; elem_list[5] = 3;
elem_list[6] = 3; elem_list[7] = 3;
elem_list[8] = 3; elem_list[9] = 3;
elem_list[10] = 3; elem_list[11] = 4;
elem_list[12] = 4; elem_list[13] = 4;
elem_list[14] = 4; elem_list[15] = 5;
elem_list[16] = 5; elem_list[17] = 5;
elem_list[18] = 5; elem_list[19] = 5;

error = ex_cvt_nodes_to_sides(exoid,
                             num_elem_per_set,
                             num_nodes_per_set,
                             elem_ind,
                             node_ind,
                             elem_list,
                             node_list,
                             side_list);

num_df_per_set[0] = 4;
num_df_per_set[1] = 4;
num_df_per_set[2] = 0;
num_df_per_set[3] = 0;
num_df_per_set[4] = 0;

df_ind[0] = 0;
df_ind[1] = 4;

dist_fact[0] = 30.0; dist_fact[1] = 30.1;
dist_fact[2] = 30.2; dist_fact[3] = 30.3;

dist_fact[4] = 31.0; dist_fact[5] = 31.1;
dist_fact[6] = 31.2; dist_fact[7] = 31.3;

error = ex_put_concat_side_sets (exoid, ids, num_elem_per_set,
                                num_df_per_set, elem_ind, df_ind,
                                elem_list, side_list, dist_fact);

END COMMENTED OUT SECTION */

error = ex_put_prop(exoid, EX_SIDE_SET, 30, "COLOR", 100);

```

```

    error = ex_put_prop(exoid, EX_SIDE_SET, 31, "COLOR", 101);
/* write QA records */
    num_qa_rec = 2;

    qa_record[0][0] = "TESTWT";
    qa_record[0][1] = "testwt";
    qa_record[0][2] = "07/07/93";
    qa_record[0][3] = "15:41:33";
    qa_record[1][0] = "FASTQ";
    qa_record[1][1] = "fastq";
    qa_record[1][2] = "07/07/93";
    qa_record[1][3] = "16:41:33";

    error = ex_put_qa (exoid, num_qa_rec, qa_record);
/* write information records */
    num_info = 3;

    info[0] = "This is the first information record.";
    info[1] = "This is the second information record.";
    info[2] = "This is the third information record.";

    error = ex_put_info (exoid, num_info, info);
/* write results variables parameters and names */
    num_glo_vars = 1;

    var_names[0] = "glo_vars";

    error = ex_put_var_param (exoid, "g", num_glo_vars);
    error = ex_put_var_names (exoid, "g", num_glo_vars, var_names);

    num_nod_vars = 2;

    var_names[0] = "nod_var0";
    var_names[1] = "nod_var1";

    error = ex_put_var_param (exoid, "n", num_nod_vars);
    error = ex_put_var_names (exoid, "n", num_nod_vars, var_names);

    num_ele_vars = 3;

    var_names[0] = "ele_var0";
    var_names[1] = "ele_var1";
    var_names[2] = "ele_var2";

    error = ex_put_var_param (exoid, "e", num_ele_vars);
    error = ex_put_var_names (exoid, "e", num_ele_vars, var_names);
/* write element variable truth table */
    truth_tab = (int *) calloc ((num_elem_blk*num_ele_vars), sizeof(int));

    k = 0;
    for (i=0; i<num_elem_blk; i++)
    {
        for (j=0; j<num_ele_vars; j++)
        {
            truth_tab[k++] = 1;
        }
    }

    error = ex_put_elem_var_tab (exoid, num_elem_blk, num_ele_vars, truth_tab);

    free (truth_tab);
/* for each time step, write the analysis results;
* the code below fills the arrays glob_var_vals,

```

```

* nodal_var_vals, and elem_var_vals with values for debugging purposes;
* obviously the analysis code will populate these arrays
*/

whole_time_step = 1;
num_time_steps = 10;

glob_var_vals = (float *) calloc (num_glo_vars, CPU_word_size);
nodal_var_vals = (float *) calloc (num_nodes, CPU_word_size);
elem_var_vals = (float *) calloc (4, CPU_word_size);

for (i=0; i<num_time_steps; i++)
{
    time_value = (float)(i+1)/100.;
/* write time value */

    error = ex_put_time (exoid, whole_time_step, &time_value);
/* write global variables */

    for (j=0; j<num_glo_vars; j++)
    {
        glob_var_vals[j] = (float)(j+2) * time_value;
    }

    error = ex_put_glob_vars (exoid, whole_time_step, num_glo_vars,
                              glob_var_vals);
/* write nodal variables */

    for (k=1; k<=num_nod_vars; k++)
    {
        for (j=0; j<num_nodes; j++)
        {
            nodal_var_vals[j] = (float)k + ((float)(j+1) * time_value);
        }

        error = ex_put_nodal_var (exoid, whole_time_step, k, num_nodes,
                                   nodal_var_vals);
    }
/* write element variables */

    for (k=1; k<=num_ele_vars; k++)
    {
        for (j=0; j<num_elem_blk; j++)
        {
            for (m=0; m<num_elem_in_block[j]; m++)
            {
                elem_var_vals[m] = (float)(k+1) + (float)(j+2) +
                                   ((float)(m+1)*time_value);
            }
            error = ex_put_elem_var (exoid, whole_time_step, k, ebids[j],
                                     num_elem_in_block[j], elem_var_vals);
        }
    }
    whole_time_step++;

/* update the data file; this should be done at the end of every time step
* to ensure that no data is lost if the analysis dies
*/
    error = ex_update (exoid);
}
free(glob_var_vals);
free(nodal_var_vals);
free(elem_var_vals);

/* close the EXODUS files
*/
error = ex_close (exoid);
}

```

## C Read Example Code

The following C program reads data from an EXODUS II file:

```
#include <stdio.h>
#include "netcdf.h"
#include "exodusII.h"

main ()
{
    int exoid, num_dim, num_nodes, num_elem, num_elem_blk, num_node_sets;
    int num_side_sets, error;
    int i, j, k, m, node_ctr;
    int *elem_map, *connect, *node_list, *node_ctr_list, *elem_list, *side_list;
    int *ids;
    int *num_sides_per_set, *num_nodes_per_set, *num_elem_per_set;
    int *num_df_per_set;
    int *node_ind, *elem_ind, *df_ind, *side_ind, num_qa_rec, num_info;
    int num_glo_vars, num_nod_vars, num_ele_vars;
    int *truth_tab;
    int whole_time_step, num_time_steps;
    int id, *num_elem_in_block, *num_nodes_per_elem, *num_attr;
    int num_nodes_in_set, num_elem_in_set;
    int num_sides_in_set, num_df_in_set;
    int list_len, elem_list_len, node_list_len, side_list_len, df_list_len;
    int node_num, time_step, var_index, beg_time, end_time, elem_num;
    int CPU_word_size, IO_word_size;
    int prop_array[2], num_props, prop_value, *prop_values;

    float *glob_var_vals, *nodal_var_vals, *elem_var_vals;
    float time_value, *time_values, *var_values;
    float *x, *y, *z, *dummy;
    float attrib[1], *dist_fact;
    float version, fdum;

    char *coord_names[3], *qa_record[2][4], *info[3], *var_names[3];
    char title[MAX_LINE_LENGTH+1], elem_type[MAX_STR_LENGTH+1];
    char *cdum;
    char *prop_names[3];

    dummy = 0; /* assign this so the Cray compiler doesn't complain */
    cdum = 0;

    CPU_word_size = 0; /* float or double */
    IO_word_size = 0; /* use what is stored in file */

    /* open EXODUS II files */

    exoid = ex_open ("test.exo", /* filename path */
                    EX_READ, /* access mode = READ */
                    &CPU_word_size, /* CPU word size */
                    &IO_word_size, /* IO word size */
                    &version); /* ExodusII library version */

    if (exoid < 0) exit(1);

    /* ncopts = NC_VERBOSE; */

    /* read database parameters */

    error = ex_get_init (exoid, title, &num_dim, &num_nodes, &num_elem,
                        &num_elem_blk, &num_node_sets, &num_side_sets);

    /* read nodal coordinates values and names from database */

    x = (float *) calloc(num_nodes, sizeof(float));
    y = (float *) calloc(num_nodes, sizeof(float));
    if (num_dim >= 3)
        z = (float *) calloc(num_nodes, sizeof(float));
    else
```

```

    z = 0;

error = ex_get_coord (exoid, x, y, z);

free (x);
free (y);
if (num_dim >= 3)
    free (z);

for (i=0; i<num_dim; i++)
{
    coord_names[i] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));
}

error = ex_get_coord_names (exoid, coord_names);

for (i=0; i<num_dim; i++)
    free(coord_names[i]);

/* read element order map */

elem_map = (int *) calloc(num_elem, sizeof(int));

error = ex_get_map (exoid, elem_map);

free (elem_map);

/* read element block parameters */

ids = (int *) calloc(num_elem_blk, sizeof(int));
num_elem_in_block = (int *) calloc(num_elem_blk, sizeof(int));
num_nodes_per_elem = (int *) calloc(num_elem_blk, sizeof(int));
num_attr = (int *) calloc(num_elem_blk, sizeof(int));

error = ex_get_elem_blk_ids (exoid, ids);

for (i=0; i<num_elem_blk; i++)
{
    error = ex_get_elem_block (exoid, ids[i], elem_type,
                              &(num_elem_in_block[i]),
                              &(num_nodes_per_elem[i]), &(num_attr[i]));
}

/* read element block properties */
error = ex_inquire (exoid, EX_INQ_EB_PROP, &num_props, &fdum, cdum);

for (i=0; i<num_props; i++)
{
    prop_names[i] = (char *) calloc ((MAX_VAR_NAME_LENGTH+1), sizeof(char));
}

error = ex_get_prop_names(exoid, EX_ELEM_BLOCK, prop_names);

for (i=0; i<num_props; i++)
{
    for (j=0; j<num_elem_blk; j++)
    {
        error = ex_get_prop(exoid, EX_ELEM_BLOCK, ids[j], prop_names[i],
                            &prop_value);
    }
}

for (i=0; i<num_props; i++)
    free(prop_names[i]);

/* read element connectivity */

for (i=0; i<num_elem_blk; i++)
{
    connect = (int *) calloc((num_nodes_per_elem[i] * num_elem_in_block[i]),
                              sizeof(int));
}

```

```

        error = ex_get_elem_conn (exoid, ids[i], connect);
        free (connect);
    }
/* read element block attributes */
    for (i=0; i<num_elem_blk; i++)
    {
        error = ex_get_elem_attr (exoid, ids[i], attrib);
    }

    free (ids);
    free (num_nodes_per_elem);
    free (num_attr);
/* read individual node sets */
    ids = (int *) calloc(num_node_sets, sizeof(int));
    error = ex_get_node_set_ids (exoid, ids);

    for (i=0; i<num_node_sets; i++)
    {
        error = ex_get_node_set_param (exoid, ids[i],
            &num_nodes_in_set, &num_df_in_set);

        node_list = (int *) calloc(num_nodes_in_set, sizeof(int));
        dist_fact = (float *) calloc(num_nodes_in_set, sizeof(float));

        error = ex_get_node_set (exoid, ids[i], node_list);

        if (num_df_in_set > 0)
        {
            error = ex_get_node_set_dist_fact (exoid, ids[i], dist_fact);
        }

        free (node_list);
        free (dist_fact);
    }
    free(ids);

/* read node set properties */
    error = ex_inquire (exoid, EX_INQ_NS_PROP, &num_props, &fdum, cdum);

    for (i=0; i<num_props; i++)
    {
        prop_names[i] = (char *) calloc ((MAX_VAR_NAME_LENGTH+1), sizeof(char));
    }
    prop_values = (int *) calloc (num_node_sets, sizeof(int));

    error = ex_get_prop_names(exoid, EX_NODE_SET, prop_names);

    for (i=0; i<num_props; i++)
    {
        error = ex_get_prop_array(exoid, EX_NODE_SET, prop_names[i],
            prop_values);
    }
    for (i=0; i<num_props; i++)
        free(prop_names[i]);
    free(prop_values);
/* read concatenated node sets; this produces the same information as
 * the above code which reads individual node sets
 */
    error = ex_inquire (exoid, EX_INQ_NODE_SETS, &num_node_sets, &fdum, cdum);

    ids = (int *) calloc(num_node_sets, sizeof(int));
    num_nodes_per_set = (int *) calloc(num_node_sets, sizeof(int));
    num_df_per_set = (int *) calloc(num_node_sets, sizeof(int));
    node_ind = (int *) calloc(num_node_sets, sizeof(int));

```



```

df_ind = (int *) calloc(num_node_sets, sizeof(int));

error = ex_inquire (exoid, EX_INQ_NS_NODE_LEN, &list_len, &fdum, cdum);
node_list = (int *) calloc(list_len, sizeof(int));

error = ex_inquire (exoid, EX_INQ_NS_DF_LEN, &list_len, &fdum, cdum);
dist_fact = (float *) calloc(list_len, sizeof(float));

error = ex_get_concat_node_sets (exoid,ids,num_nodes_per_set,num_df_per_set,
                                node_ind, df_ind, node_list, dist_fact);

free (ids);
free (num_nodes_per_set);
free (df_ind);
free (node_ind);
free (num_df_per_set);
free (node_list);
free (dist_fact);

/* read individual side sets */

ids = (int *) calloc(num_side_sets, sizeof(int));

error = ex_get_side_set_ids (exoid, ids);

for (i=0; i<num_side_sets; i++)
{
    error = ex_get_side_set_param (exoid, ids[i], &num_sides_in_set,
                                  &num_df_in_set);

    /* Note: The # of elements is same as # of sides! */
    num_elem_in_set = num_sides_in_set;
    elem_list = (int *) calloc(num_elem_in_set, sizeof(int));
    side_list = (int *) calloc(num_sides_in_set, sizeof(int));
    node_ctr_list = (int *) calloc(num_elem_in_set, sizeof(int));
    node_list = (int *) calloc(num_elem_in_set*21, sizeof(int));
    dist_fact = (float *) calloc(num_df_in_set, sizeof(float));

    error = ex_get_side_set (exoid, ids[i], elem_list, side_list);

    error = ex_get_side_set_node_list (exoid, ids[i], node_ctr_list,
                                       node_list);

    if (num_df_in_set > 0)
    {
        error = ex_get_side_set_dist_fact (exoid, ids[i], dist_fact);
    }

    free (elem_list);
    free (side_list);
    free (node_ctr_list);
    free (node_list);
    free (dist_fact);
}

/* read side set properties */

error = ex_inquire (exoid, EX_INQ_SS_PROP, &num_props, &fdum, cdum);

for (i=0; i<num_props; i++)
{
    prop_names[i] = (char *) calloc ((MAX_VAR_NAME_LENGTH+1), sizeof(char));
}

error = ex_get_prop_names(exoid,EX_SIDE_SET,prop_names);

for (i=0; i<num_props; i++)
{
    for (j=0; j<num_side_sets; j++)
    {
        error = ex_get_prop(exoid, EX_SIDE_SET, ids[j], prop_names[i],
                            &prop_value);
    }
}

```

```

    }
}
for (i=0; i<num_props; i++)
    free(prop_names[i]);
free (ids);

error = ex_inquire (exoid, EX_INQ_SIDE_SETS, &num_side_sets, &fdum, cdum);

if (num_side_sets > 0)
{
    error = ex_inquire(exoid, EX_INQ_SS_ELEM_LEN, &elem_list_len, &fdum, cdum);

    error = ex_inquire(exoid, EX_INQ_SS_NODE_LEN, &node_list_len, &fdum, cdum);

    error = ex_inquire(exoid, EX_INQ_SS_DF_LEN, &df_list_len, &fdum, cdum);
}

/* read concatenated side sets; this produces the same information as
 * the above code which reads individual side sets
 */

/* concatenated side set read */

ids = (int *) calloc(num_side_sets, sizeof(int));
num_elem_per_set = (int *) calloc(num_side_sets, sizeof(int));
num_df_per_set = (int *) calloc(num_side_sets, sizeof(int));
elem_ind = (int *) calloc(num_side_sets, sizeof(int));
df_ind = (int *) calloc(num_side_sets, sizeof(int));
elem_list = (int *) calloc(elem_list_len, sizeof(int));
side_list = (int *) calloc(elem_list_len, sizeof(int));
dist_fact = (float *) calloc(df_list_len, sizeof(float));

error = ex_get_concat_side_sets (exoid, ids, num_elem_per_set,
                                num_df_per_set, elem_ind, df_ind,
                                elem_list, side_list, dist_fact);

free (ids);
free (num_elem_per_set);
free (num_df_per_set);
free (df_ind);
free (elem_ind);
free (elem_list);
free (side_list);
free (dist_fact);

/* end of concatenated side set read */

/* read QA records */

ex_inquire (exoid, EX_INQ_QA, &num_qa_rec, &fdum, cdum);

for (i=0; i<num_qa_rec; i++)
{
    for (j=0; j<4; j++)
    {
        qa_record[i][j] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));
    }
}

error = ex_get_qa (exoid, qa_record);

/* read information records */

error = ex_inquire (exoid, EX_INQ_INFO, &num_info, &fdum, cdum);

for (i=0; i<num_info; i++)
{
    info[i] = (char *) calloc ((MAX_LINE_LENGTH+1), sizeof(char));
}
error = ex_get_info (exoid, info);
for (i=0; i<num_info; i++)
{

```

```

    free(info[i]);
}
/* read global variables parameters and names */
error = ex_get_var_param (exoid, "g", &num_glo_vars);
for (i=0; i<num_glo_vars; i++)
{
    var_names[i] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));
}
error = ex_get_var_names (exoid, "g", num_glo_vars, var_names);
for (i=0; i<num_glo_vars; i++)
{
    free(var_names[i]);
}
/* read nodal variables parameters and names */
error = ex_get_var_param (exoid, "n", &num_nod_vars);
for (i=0; i<num_nod_vars; i++)
{
    var_names[i] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));
}
error = ex_get_var_names (exoid, "n", num_nod_vars, var_names);
for (i=0; i<num_nod_vars; i++)
{
    free(var_names[i]);
}
/* read element variables parameters and names */
error = ex_get_var_param (exoid, "e", &num_ele_vars);
for (i=0; i<num_ele_vars; i++)
{
    var_names[i] = (char *) calloc ((MAX_STR_LENGTH+1), sizeof(char));
}
error = ex_get_var_names (exoid, "e", num_ele_vars, var_names);
for (i=0; i<num_ele_vars; i++)
{
    free(var_names[i]);
}
/* read element variable truth table */
truth_tab = (int *) calloc ((num_elem_blk*num_ele_vars), sizeof(int));
error = ex_get_elem_var_tab (exoid, num_elem_blk, num_ele_vars, truth_tab);
free (truth_tab);
/* determine how many time steps are stored */
error = ex_inquire (exoid, EX_INQ_TIME, &num_time_steps, &fdum, cdum);
/* read time value at one time step */
time_step = 3;
error = ex_get_time (exoid, time_step, &time_value);
/* read time values at all time steps */
time_values = (float *) calloc (num_time_steps, sizeof(float));

```

```

    error = ex_get_all_times (exoid, time_values);
    free (time_values);

/* read all global variables at one time step */

    var_values = (float *) calloc (num_glo_vars, sizeof(float));

    error = ex_get_glob_vars (exoid, time_step, num_glo_vars, var_values);
    free (var_values);

/* read a single global variable through time */

    var_index = 1;
    beg_time = 1;
    end_time = -1;

    var_values = (float *) calloc (num_time_steps, sizeof(float));

    error = ex_get_glob_var_time (exoid, var_index, beg_time, end_time,
                                var_values);
    free (var_values);

/* read a nodal variable at one time step */

    var_values = (float *) calloc (num_nodes, sizeof(float));

    error = ex_get_nodal_var (exoid, time_step, var_index, num_nodes,
                             var_values);

    free (var_values);

/* read a nodal variable through time */

    var_values = (float *) calloc (num_time_steps, sizeof(float));

    node_num = 1;
    error = ex_get_nodal_var_time (exoid, var_index, node_num, beg_time,
                                  end_time, var_values);

    free (var_values);

/* read an element variable at one time step */

    ids = (int *) calloc(num_elem_blk, sizeof(int));

    error = ex_get_elem_blk_ids (exoid, ids);

    for (i=0; i<num_elem_blk; i++)
    {
        var_values = (float *) calloc (num_elem_in_block[i], sizeof(float));

        error = ex_get_elem_var (exoid, time_step, var_index, ids[i],
                                num_elem_in_block[i], var_values);

        free (var_values);
    }
    free (num_elem_in_block);
    free(ids);

/* read an element variable through time */

    var_values = (float *) calloc (num_time_steps, sizeof(float));

    var_index = 2;
    elem_num = 2;
    error = ex_get_elem_var_time (exoid, var_index, elem_num, beg_time,
                                  end_time, var_values);

    free (var_values);

    error = ex_close (exoid);
}

```

## FORTRAN Write Example Code

The following Fortran program creates an EXODUS II file and populates it. Although this sample code does not conform entirely to the ANSI Fortran-77 standard (i.e., lengths of variable names, included files, etc.), it has successfully compiled and executed on all UNIX workstations we have attempted and is included only as an example.

```
      program testwt
c
c This is a test program for the Fortran binding of the EXODUS II
c database write routines.
c
      include 'exodusII.inc'

      integer iin, iout
      integer exoid, num_dim, num_nodes, num_elem, num_elem_blk
      integer num_elem_in_block(2), num_node_sets
      integer num_side_sets
      integer i, j, k, m, elem_map(2), connect(4)
      integer node_list(10), elem_list(10), side_list(10)
      integer ebids(2), ids(2), num_nodes_per_set(2), num_elem_per_set(2)
      integer num_df_per_set(2)
      integer df_ind(2), node_ind(2), elem_ind(2), num_qa_rec, num_info
      integer num_glo_vars, num_nod_vars, num_ele_vars
      integer truth_tab(3,2)
      integer whole_time_step, num_time_steps
      integer cpu_word_size, io_word_size
      integer prop_array(2)

      real glob_var_vals(10), nodal_var_vals(8)
      real time_value, elem_var_vals(20)
      real x(8), y(8), dummy(1)
      real attrib(1), dist_fact(8)

      character*(MXSTLN) coord_names(3)
      character*(MXSTLN) cname
      character*(MXSTLN) var_names(3)
      character*(MXSTLN) qa_record(4,2)
      character*(MXLNLN) inform(3)
      character*(MXSTLN) prop_names(2)

      data iin /5/, iout /6/

      cpu_word_size = 0
      io_word_size = 0
c
c create EXODUS II files
c
      exoid = excre ("test.exo",
1          EXCLOB, cpu_word_size, io_word_size, ierr)
c
c initialize file with parameters
c

      num_dim = 2
      num_nodes = 8
      num_elem = 2
      num_elem_blk = 2
      num_node_sets = 2
      num_side_sets = 2

      call expini (exoid, "This is a test", num_dim, num_nodes,
1          num_elem, num_elem_blk, num_node_sets,
2          num_side_sets, ierr)
```

```

c
c write nodal coordinates values and names to database
c
      x(1) = 0.0
      x(2) = 1.0
      x(3) = 1.0
      x(4) = 0.0
      x(5) = 1.0
      x(6) = 2.0
      x(7) = 2.0
      x(8) = 1.0
      y(1) = 0.0
      y(2) = 0.0
      y(3) = 1.0
      y(4) = 1.0
      y(5) = 0.0
      y(6) = 0.0
      y(7) = 1.0
      y(8) = 1.0

      call expcor (exoid, x, y, dummy, ierr)

      coord_names(1) = "xcoor"
      coord_names(2) = "ycoor"

      call expcon (exoid, coord_names, ierr)

c
c write element order map
c
      do 10 i = 1, num_elem
          elem_map(i) = i
10      continue

      call expmap (exoid, elem_map, ierr)

c
c write element block parameters
c
      num_elem_in_block(1) = 1
      num_elem_in_block(2) = 1

      ebids(1) = 10
      ebids(2) = 11

      cname = "QUAD"

      call expelb (exoid, ebids(1), cname, num_elem_in_block(1), 4, 1, ierr)
      call expelb (exoid, ebids(2), cname, num_elem_in_block(2), 4, 1, ierr)

c write element block properties

      prop_names(1) = "TOP"
      prop_names(2) = "RIGHT"
      call exppn(exoid, EXEBLK, 2, prop_names, ierr)

      call expp(exoid, EXEBLK, ebids(1), "TOP", 1, ierr)
      call expp(exoid, EXEBLK, ebids(2), "RIGHT", 1, ierr)

c
c write element connectivity
c
      connect(1) = 1
      connect(2) = 2
      connect(3) = 3
      connect(4) = 4

      call expelc (exoid, ebids(1), connect, ierr)

```

```

connect(1) = 5
connect(2) = 6
connect(3) = 7
connect(4) = 8

call expelc (exoid, ebids(2), connect, ierr)

c
c write element block attributes
c

attrib(1) = 3.14159
call expeat (exoid, ebids(1), attrib, ierr)

attrib(1) = 6.14159
call expeat (exoid, ebids(2), attrib, ierr)

c
c write individual node sets
c

node_list(1) = 100
node_list(2) = 101
node_list(3) = 102
node_list(4) = 103
node_list(5) = 104

dist_fact(1) = 1.0
dist_fact(2) = 2.0
dist_fact(3) = 3.0
dist_fact(4) = 4.0
dist_fact(5) = 5.0

call expnp (exoid, 20, 5, 5, ierr)
call expns (exoid, 20, node_list, ierr)
call expnsd (exoid, 20, dist_fact, ierr)

node_list(1) = 200
node_list(2) = 201
node_list(3) = 202

dist_fact(1) = 1.1
dist_fact(2) = 2.1
dist_fact(3) = 3.1

call expnp (exoid, 21, 3, 3, ierr)
call expns (exoid, 21, node_list, ierr)
call expnsd (exoid, 21, dist_fact, ierr)

c
c write concatenated node sets; this produces the same information as
c the above code which writes individual node sets
c

ids(1) = 20
ids(2) = 21

num_nodes_per_set(1) = 5
num_nodes_per_set(2) = 3

num_df_per_set(1) = 5
num_df_per_set(2) = 3

node_ind(1) = 1
node_ind(2) = 6

df_ind(1) = 1
df_ind(2) = 6

node_list(1) = 100
node_list(2) = 101
node_list(3) = 102

```

```

node_list(4) = 103
node_list(5) = 104
node_list(6) = 200
node_list(7) = 201
node_list(8) = 202

dist_fact(1) = 1.0
dist_fact(2) = 2.0
dist_fact(3) = 3.0
dist_fact(4) = 4.0
dist_fact(5) = 5.0
dist_fact(6) = 1.1
dist_fact(7) = 2.1
dist_fact(8) = 3.1

c commented out because individual node sets already written
c   call expcns (exoid, ids, num_nodes_per_set, num_df_per_set,
c   1         node_ind, df_ind, node_list, dist_fact, ierr)

c   write node set properties

prop_names(1) = "FACE"
call expp(exoid, EXNSET, 20, prop_names(1), 4, ierr)

call expp(exoid, EXNSET, 21, prop_names(1), 5, ierr)

prop_array(1) = 1000
prop_array(2) = 2000

prop_names(1) = "FRONT"
call exppa(exoid, EXNSET, prop_names(1), prop_array, ierr)

c
c write individual side sets
c

elem_list(1) = 11
elem_list(2) = 12

side_list(1) = 1
side_list(2) = 2

dist_fact(1) = 30.0
dist_fact(2) = 30.1
dist_fact(3) = 30.2
dist_fact(4) = 30.3

call expsp (exoid, 30, 2, 4, ierr)
call expss (exoid, 30, elem_list, side_list, ierr)
call expssd (exoid, 30, dist_fact, ierr)

elem_list(1) = 13
elem_list(2) = 14

side_list(1) = 3
side_list(2) = 4

dist_fact(1) = 31.0
dist_fact(2) = 31.1
dist_fact(3) = 31.2
dist_fact(4) = 31.3

call expsp (exoid, 31, 2, 4, ierr)
call expss (exoid, 31, elem_list, side_list, ierr)
call expssd (exoid, 31, dist_fact, ierr)

c write concatenated side sets; this produces the same information as
c the above code which writes individual side sets
c

ids(1) = 30
ids(2) = 31

```



```

num_elem_per_set(1) = 2
num_elem_per_set(2) = 2

num_df_per_set(1) = 4
num_df_per_set(2) = 4

elem_ind(1) = 1
elem_ind(2) = 3

df_ind(1) = 1
df_ind(2) = 5

elem_list(1) = 11
elem_list(2) = 12
elem_list(3) = 13
elem_list(4) = 14

side_list(1) = 1
side_list(2) = 2
side_list(3) = 3
side_list(4) = 4

dist_fact(1) = 30.0
dist_fact(2) = 30.1
dist_fact(3) = 30.2
dist_fact(4) = 30.3
dist_fact(5) = 31.0
dist_fact(6) = 31.1
dist_fact(7) = 31.2
dist_fact(8) = 31.3

c commented out because individual side sets already written
c   call expcss (exoid, ids, num_elem_per_set, num_df_per_set,
c   1           elem_ind, df_ind, elem_list, side_list, dist_fact,
c   2           ierr)

prop_names(1) = "COLOR"
call expp(exoid, EXSSET, 30, prop_names(1), 100, ierr)

call expp(exoid, EXSSET, 31, prop_names(1), 101, ierr)
c
c write QA records
c

num_qa_rec = 2

qa_record(1,1) = "TESTWT fortran version"
qa_record(2,1) = "testwt"
qa_record(3,1) = "07/07/93"
qa_record(4,1) = "15:41:33"
qa_record(1,2) = "FASTQ"
qa_record(2,2) = "fastq"
qa_record(3,2) = "07/07/93"
qa_record(4,2) = "16:41:33"

call expqa (exoid, num_qa_rec, qa_record, ierr)

c
c write information records
c

num_info = 3

inform(1) = "This is the first information record."
inform(2) = "This is the second information record."
inform(3) = "This is the third information record."

call expinf (exoid, num_info, inform, ierr)

```

```

c
c write results variables parameters and names
c
    num_glo_vars = 1

    var_names(1) = "glo_vars"

    call expvp (exoid, "g", num_glo_vars, ierr)
    call expvan (exoid, "g", num_glo_vars, var_names, ierr)

    num_nod_vars = 2

    var_names(1) = "nod_var0"
    var_names(2) = "nod_var1"

    call expvp (exoid, "n", num_nod_vars, ierr)
    call expvan (exoid, "n", num_nod_vars, var_names, ierr)

    num_ele_vars = 3

    var_names(1) = "ele_var0"
    var_names(2) = "ele_var1"
    var_names(3) = "ele_var2"

    call expvp (exoid, "e", num_ele_vars, ierr)
    call expvan (exoid, "e", num_ele_vars, var_names, ierr)

c
c write element variable truth table
c
    k = 0

    do 30 i = 1,num_elem_blk
        do 20 j = 1,num_ele_vars
            truth_tab(j,i) = 1
20        continue
30    continue
    call expvtt (exoid, num_elem_blk, num_ele_vars, truth_tab,ierr)

c
c for each time step, write the analysis results;
c the code below fills the arrays glob_var_vals,
c nodal_var_vals, and elem_var_vals with values for debugging purposes;
c obviously the analysis code will populate these arrays
c
    whole_time_step = 1
    num_time_steps = 10

    do 110 i = 1, num_time_steps
        time_value = real(i)/100.
c
c write time value
c
        call exptime (exoid, whole_time_step, time_value, ierr)

c
c write global variables
c
    do 50 j = 1, num_glo_vars
        glob_var_vals(j) = real(j+1) * time_value
50    continue

    call expgv (exoid, whole_time_step, num_glo_vars,
1        glob_var_vals, ierr)

```

```

c
c write nodal variables
c

      do 70 k = 1, num_nod_vars
        do 60 j = 1, num_nodes

          nodal_var_vals(j) = real(k) + (real(j) * time_value)

60      continue

          call expnv (exoid, whole_time_step, k, num_nodes,
1          nodal_var_vals, ierr)
70      continue

c
c write element variables
c

      do 100 k = 1, num_ele_vars
        do 90 j = 1, num_elem_blk
          do 80 m = 1, num_elem_in_block(j)

            elem_var_vals(m) = real(k+1) + real(j+1) +
1            (real(m)*time_value)

80      continue

            call expev (exoid, whole_time_step, k, ebids(j),
1            num_elem_in_block(j), elem_var_vals, ierr)

90      continue
100     continue

      whole_time_step = whole_time_step + 1

c
c update the data file; this should be done at the end of every time
c step to ensure that no data is lost if the analysis dies
c

      call exupda (exoid, ierr)

110    continue

c
c close the EXODUS files
c

      call exclos (exoid, ierr)

      stop
      end

```

## FORTRAN Read Example Code

The following Fortran program reads data from an EXODUS II file:

```
program testrd

c
c This is a test program for the Fortran binding of the EXODUS II
c database read routines
c
  implicit none

  include 'exodusII.inc'

  integer iin, iout, ierr
  integer exoid, num_dim, num_nodes, num_elem, num_elem_blk
  integer num_node_sets
  integer num_side_sets
  integer i, j, elem_map(2), connect(4), node_list(10)
  integer elem_list(10), side_list(10), ids(5)
  integer num_elem_per_set(2), num_nodes_per_set(2)
  integer num_df_per_set(2)
  integer num_df_in_set, num_sides_in_set
  integer df_ind(2), node_ind(2), elem_ind(2), num_qa_rec, num_info
  integer num_glo_vars, num_nod_vars, num_ele_vars
  integer truth_tab(3,2)
  integer num_time_steps
  integer num_elem_in_block(2), num_nodes_per_elem(2)
  integer num_attr(2)
  integer num_nodes_in_set, num_elem_in_set
  integer df_list_len, list_len, elem_list_len
  integer node_num, time_step, var_index, beg_time, end_time
  integer elem_num
  integer cpu_ws, io_ws
  integer num_props, prop_value

  real time_value, time_values(10), var_values(10)
  real x(8), y(8), dummy(1)
  real attrib(1), dist_fact(8)
  real vers, fdum

  character*(MXSTLN) coord_names(3), qa_record(4,2), var_names(3)
  character*(MXLNLN) inform(3), titl
  character typ*(MXSTLN), cdum*1
  character*(MXSTLN) prop_names(3)

  data iin /5/, iout /6/

c
c open EXODUS II files
c
  cpu_ws = 0
  io_ws = 0

  exoid = exopen ("test.exo", EXREAD, cpu_ws, io_ws, vers, ierr)

c
c read database parameters
c
  call exgini (exoid, titl, num_dim, num_nodes, num_elem,
1             num_elem_blk, num_node_sets, num_side_sets, ierr)

c
c read nodal coordinates values and names from database
c
  call exgcor (exoid, x, y, dummy, ierr)
```

```

        call exgcon (exoid, coord_names, ierr)

c
c read element order map
c
        call exgmap (exoid, elem_map, ierr)

c
c read element block parameters
c
c
        call exgebi (exoid, ids, ierr)

        do 40 i = 1, num_elem_blk
            call exgelb (exoid, ids(i), typ, num_elem_in_block(i),
1             num_nodes_per_elem(i), num_attr(i), ierr)
40    continue

c    read element block properties */
        call exinq (exoid, EXNEBP, num_props, fdum, cdum, ierr)
        call exgpn(exoid, EXEBLK, prop_names, ierr)

        do 47 i = 1, num_props
            do 45 j = 1, num_elem_blk
                call exgp(exoid, EXEBLK, ids(j), prop_names(i), prop_value, ierr)
45    continue
47    continue

c
c read element connectivity
c
        do 60 i = 1, num_elem_blk
            call exgelc (exoid, ids(i), connect, ierr)

60    continue

c
c read element block attributes
c
        do 70 i = 1, num_elem_blk
            call exgeat (exoid, ids(i), attrib, ierr)

70    continue

c
c read individual node sets
c
        if (num_node_sets .gt. 0) then
            call exgnsi (exoid, ids, ierr)
        endif

        do 100 i = 1, num_node_sets
            call exgnp (exoid, ids(i), num_nodes_in_set,
1             num_df_in_set, ierr)

            call exgns (exoid, ids(i), node_list, ierr)
            call exgnsd (exoid, ids(i), dist_fact, ierr)

100    continue

```

```

c   read node set properties

   call exinq (exoid, EXNNSP, num_props, fdum, cdum, ierr)

   call exgpn(exoid, EXNSET, prop_names, ierr)

   do 107 i = 1, num_props
     do 105 j = 1, num_node_sets
       call exgp(exoid,EXNSET,ids(j),prop_names(i),prop_value,ierr)
105     continue
107   continue

c
c read concatenated node sets; this produces the same information as
c the above code which reads individual node sets
c
   call exinq (exoid, EXNODS, num_node_sets, fdum, cdum, ierr)

   if (num_node_sets .gt. 0) then
     call exinq (exoid, EXNSNL, list_len, fdum, cdum, ierr)

     call exinq (exoid, EXNSDF, list_len, fdum, cdum, ierr)

     call exgcns (exoid, ids, num_nodes_per_set, num_df_per_set,
1     node_ind, df_ind, node_list, dist_fact, ierr)

   endif

c
c read individual side sets
c
   if (num_side_sets .gt. 0) then
     call exgssi (exoid, ids, ierr)
   endif

   do 190 i = 1, num_side_sets
     call exgsp (exoid, ids(i), num_sides_in_set, num_df_in_set,
1     ierr)

     call exgss (exoid, ids(i), elem_list, side_list, ierr)

     call exgssd (exoid, ids(i), dist_fact, ierr)

     num_elem_in_set = num_sides_in_set

190   continue

c   read side set properties

   call exinq (exoid, EXNSSP, num_props, fdum, cdum, ierr)

   call exgpn(exoid, EXSSET, prop_names, ierr)

   do 197 i = 1, num_props
     do 195 j = 1, num_side_sets
       call exgp(exoid, EXSSET,ids(j),prop_names(i),prop_value,ierr)
195     continue
197   continue

   call exinq (exoid, EXSIDS, num_side_sets, fdum, cdum, ierr)

   if (num_side_sets .gt. 0) then
     call exinq (exoid, EXSSEL, elem_list_len, fdum, cdum, ierr)
     call exinq (exoid, EXSSDF, df_list_len, fdum, cdum, ierr)

c
c read concatenated side sets; this produces the same information as
c the above code which reads individual side sets
c
   call exgcss (exoid, ids, num_elem_per_set, num_df_per_set,
1     elem_ind, df_ind, elem_list, side_list, dist_fact,
2     ierr)
   endif

```

```

c
c read QA records
c
    call exinq (exoid, EXQA, num_qa_rec, fdum, cdum, ierr)
    call exgqa (exoid, qa_record, ierr)

c
c read information records
c
    call exinq (exoid, EXINFO, num_info, fdum, cdum, ierr)
    call exginf (exoid, inform, ierr)

c
c read global variables parameters and names
c
    call exgvp (exoid, "g", num_glo_vars, ierr)
    call exgvan (exoid, "g", num_glo_vars, var_names, ierr)

c
c read nodal variables parameters and names
c
    call exgvp (exoid, "n", num_nod_vars, ierr)
    call exgvan (exoid, "n", num_nod_vars, var_names, ierr)

c
c read element variables parameters and names
c
    call exgvp (exoid, "e", num_ele_vars, ierr)
    call exgvan (exoid, "e", num_ele_vars, var_names, ierr)

c
c read element variable truth table
c
    call exgvtt (exoid, num_elem_blk, num_ele_vars, truth_tab, ierr)

c
c determine how many time steps are stored
c
    call exinq (exoid, EXTIMS, num_time_steps, fdum, cdum, ierr)

c
c read time value at one time step
c
    time_step = 3
    call exgtim (exoid, time_step, time_value, ierr)

c
c read time values at all time steps
c
    call exgatm (exoid, time_values, ierr)

    var_index = 1
    beg_time = 1
    end_time = -1

c
c read all global variables at one time step
c
    call exggv (exoid, time_step, num_glo_vars, var_values, ierr)

c
c read a single global variable through time
c
    call exggvt (exoid, var_index, beg_time, end_time, var_values,
1               ierr)

```

```

c
c read a nodal variable at one time step
c
    call exgnv (exoid, time_step, var_index, num_nodes, var_values,
1             ierr)

c
c read a nodal variable through time
c
    node_num = 1

    call exgnvt (exoid, var_index, node_num, beg_time, end_time,
1             var_values, ierr)

c
c read an element variable at one time step
c
    call exgebi (exoid, ids, ierr)

    do 450 i = 1, num_elem_blk

        call exgev (exoid, time_step, var_index, ids(i),
1             num_elem_in_block(i), var_values, ierr)

450    continue

c
c read an element variable through time
c
    var_index = 2
    elem_num = 2

    call exgevt (exoid, var_index, elem_num, beg_time, end_time,
1             var_values, ierr)

    call exclos (exoid, ierr)

    stop
end

```



# Index

## A

access mode 25, 26  
ASCII version of file 22  
attributes  
    description 9  
    number of 9, 65, 66, 68, 69  
    read 75  
    table of 9  
    write 73

## B

boundary conditions 11, 12

## C

character strings 5  
clobber mode 23, 24  
close file 27  
concatenated node sets  
    description 11  
    read 91  
    write 87  
concatenated side sets  
    description 15  
    read 112  
    write 108  
connectivity array 71, 72  
conventions  
    general 4  
    node ordering 9  
convert  
    netCDF to ASCII 22  
    node list to side list 115  
coordinate names  
    description 6  
    read 55  
    write 53  
coordinates, see nodal coordinates  
current 28

## D

dimensionality 5, 29, 30, 31, 32  
distribution factors, see side set; node set

## E

efficiency issues A-1

element attributes, see attributes  
element block  
    description 8  
    ID 65, 66, 68, 69  
    ID, description 8  
    number of 5, 29, 30, 31, 32  
    parameters, read 68  
    parameters, write 65  
    read IDs 70, 128  
element connectivity  
    description 9  
    read 72  
    write 71  
element number map  
    description 7  
    read 61  
    write 59  
element numbering 7, 8  
element order map 8, 62, 64  
element results, see element variable  
element topology 9  
element type 8, 65, 66, 68, 69  
    table of 9  
element variable 20  
    description 4  
    read values 152, 154  
    truth table 20  
    truth table, read 147  
    truth table, description 20  
    truth table, write 145  
    write values 149  
elements  
    number of 5, 29, 30, 31, 32  
error level 47  
error messages C-1  
error reporting 45  
ex\_close 27, B-1  
ex\_create 23, B-1  
ex\_cvt\_nodes\_to\_sides 115, B-1  
ex\_err 45, B-1  
ex\_get\_all\_times 19, 143, B-1  
ex\_get\_concat\_node\_sets 11, 91, B-1  
ex\_get\_concat\_side\_sets 15, 112, B-2

ex\_get\_coord 6, 51, B-2  
 ex\_get\_coord\_names 6, 55, B-2  
 ex\_get\_elem\_attr 9, 75, B-2  
 ex\_get\_elem\_blk\_ids 8, 70, B-2  
 ex\_get\_elem\_block 8, 68, B-2  
 ex\_get\_elem\_conn 9, 72, B-2  
 ex\_get\_elem\_num\_map 7, B-2  
 ex\_get\_elem\_var 20, 152, B-2  
 ex\_get\_elem\_var\_tab 20, 147, B-3  
 ex\_get\_elem\_var\_time 20, 154, B-3  
 ex\_get\_glob\_var\_time 19, 161, B-3  
 ex\_get\_glob\_vars 19, 159, B-3  
 ex\_get\_info 6, 39, B-3  
 ex\_get\_init 5, 31, B-3  
 ex\_get\_map 8, 64, B-3  
 ex\_get\_nodal\_var 19, 165, B-3  
 ex\_get\_nodal\_var\_time 19, 167, B-3  
 ex\_get\_node\_num\_map 7, 58, 61, B-4  
 ex\_get\_node\_set 11, 82, B-4  
 ex\_get\_node\_set\_dist\_fact 11, 85, B-4  
 ex\_get\_node\_set\_ids 11, 86, B-4  
 ex\_get\_node\_set\_param 11, 79, B-4  
 ex\_get\_prop 18, 124, B-4  
 ex\_get\_prop\_array 18, 128, B-4  
 ex\_get\_prop\_names 18, 120, B-4  
 ex\_get\_qa 6, 35, B-4  
 ex\_get\_side\_set 14, 100, B-4  
 ex\_get\_side\_set\_dist\_fact 15, 104, B-4  
 ex\_get\_side\_set\_ids 14, 105, B-4  
 ex\_get\_side\_set\_node\_list 14, 15, 106, B-5  
 ex\_get\_side\_set\_param 14, 96, B-5  
 ex\_get\_time 19, 141, B-5  
 ex\_get\_var\_names 19, 137, B-5  
 ex\_get\_var\_param 19, 133, B-5  
 ex\_inquire 41, B-5  
 ex\_open 25, B-5  
 ex\_opts 47, B-5  
 ex\_put\_concat\_node\_sets 11, 87, B-5  
 ex\_put\_concat\_side\_sets 15, 108, B-6  
 ex\_put\_coord 6, 49, B-6  
 ex\_put\_coord\_names 6, 53, B-6  
 ex\_put\_elem\_attr 9, 73, B-6  
 ex\_put\_elem\_block 8, 65, B-6  
 ex\_put\_elem\_conn 9, 71, B-6  
 ex\_put\_elem\_num\_map 7, B-6  
 ex\_put\_elem\_var 20, 149, B-6  
 ex\_put\_elem\_var\_tab 20, 145, B-6  
 ex\_put\_glob\_vars 19, 157, B-6  
 ex\_put\_info 6, 37, B-7  
 ex\_put\_init 5, 29, B-7  
 ex\_put\_map 8, 62, B-7  
 ex\_put\_nodal\_var 19, 163, B-7  
 ex\_put\_node\_num\_map 7, 56, 59, B-7  
 ex\_put\_node\_set 11, 81, B-7  
 ex\_put\_node\_set\_dist\_fact 11, 83, B-7  
 ex\_put\_node\_set\_param 11, 77, B-7  
 ex\_put\_prop 18, 122, B-7  
 ex\_put\_prop\_array 18, 126, B-7  
 ex\_put\_prop\_param 18, 118, B-8  
 ex\_put\_qa 6, 33, B-8  
 ex\_put\_side\_set 14, 98, B-8  
 ex\_put\_side\_set\_dist\_fact 15, 102, B-8  
 ex\_put\_side\_set\_param 14, 94, B-8  
 ex\_put\_time 19, 139, B-8  
 ex\_put\_var\_names 19, 135, B-8  
 ex\_put\_var\_param 19, 131, B-8  
 ex\_update 22, 28, B-8  
 examples  
     C code D-1, D-10  
     Fortran code D-17, D-24  
 EXCLOS 27, B-9  
 EXCN2S 116, B-9  
 EXCRE 24, B-9  
 EXERR 46, B-9  
 EXGATM 19, 144, B-9  
 EXGCNS 11, 92, B-9  
 EXGCON 6, 55, B-9  
 EXGCOR 6, 52, B-9  
 EXGCSS 15, 113, B-10  
 EXGEAT 9, 75, B-10  
 EXGEBI 8, 70, B-10  
 EXGELB 8, 69, B-10  
 EXGELC 9, 72, B-10  
 EXGENM 7, B-10  
 EXGEV 20, 153, B-10  
 EXGEVT 20, 155, B-11  
 EXGGV 19, 160, B-11  
 EXGGVT 19, 162, B-11  
 EXGINF 6, 39, B-11  
 EXGINI 5, 32, B-11  
 EXGMAP 8, 58, 61, 64, B-11  
 EXGNNM 7, B-11

EXGNP 11, 80, B-11  
EXGNS 11, 82, B-12  
EXGNSD 11, 85, B-12  
EXGNSI 11, 86, B-12  
EXGNV 19, 166, B-12  
EXGNVT 19, 168, B-12  
EXGP 18, 124, B-12  
EXGPA 18, 129, B-12  
EXGPN 18, 121, B-12  
EXGQA 6, 36, B-12  
EXGSP 14, 97, B-13  
EXGSS 14, 100, B-13  
EXGSSD 15, 104, B-13  
EXGSSI 14, 105, B-13  
EXGSSN 14, 15, 107, B-13  
EXGTIM 19, 141, B-13  
EXGVAN 19, 138, B-13  
EXGVP 19, 133, B-13  
EXGVTT 20, 148, B-13  
EXINQ 42, B-14  
EXOPEN 26, B-14  
EXOPTS 47, B-14  
EXPCNS 11, 88, B-14  
EXPCON 6, 53, B-14  
EXPCOR 6, 50, B-14  
EXPCSS 15, 110, B-14  
EXPEAT 9, 73, B-15  
EXPELB 8, 66, B-15  
EXPELC 9, 71, B-15  
EXPENM 7, B-15  
EXPEV 20, 150, B-15  
EXPGV 19, 158, B-15  
EXPINF 6, 37, B-15  
EXPINI 5, 30, B-15  
EXPMAP 8, 62, B-16  
EXPNNM 7, 56, 59, B-16  
EXPNP 11, 78, B-16  
EXPNS 11, 81, B-16  
EXPNSD 11, 83, B-16  
EXPNV 19, 164, B-16  
EXPP 18, 123, B-16  
EXPPA 18, 127, B-16  
EXPPN 18, 119, B-17  
EXPQA 6, 34, B-17  
EXPSP 14, 95, B-17  
EXPSS 14, 98, B-17

EXPSSD 15, 102, B-17  
EXPTIM 19, 139, B-17  
EXPVAN 19, 136, B-17  
EXPVP 19, 132, B-17  
EXPVTT 20, 146, B-17  
EXUPDA 22, 28, B-17

## **F**

flush buffers 22, 28  
function call summary B-1

## **G**

global variable  
    description 4, 19  
    read values 159, 161  
    write values 157

## **I**

information data  
    description 6  
    read 39  
    write 37  
initialization 4, 29, 31  
inquire parameters 41

## **M**

model 4, 48

## **N**

ncdump 22  
netCDF 3, 15, 22  
    implementation of EXODUS A-1  
nodal coordinates  
    description 6  
    read 51  
    write 49  
nodal variable  
    description 4, 19  
    read values 165, 167  
    write values 163  
node number map 58  
    description 7  
    write 56  
node numbering 7  
node ordering 9

node set  
concatenated 11, 87, 91  
description 11  
distribution factors 11, 77, 79, 83, 85  
distribution factors,description 11  
ID 11  
node list 11  
number of 5, 29, 30, 31, 32  
read 79, 82  
read IDs 86, 128  
write 77, 81

nodes  
number of 5, 29, 30, 31, 32

## O

object property  
array 118, 126, 128  
description 17  
names 118, 120  
value 18, 122, 124

open file 25

## Q

quality assurance (QA) records  
description 6  
read 35  
write 33

query database 41

## R

results  
data description 19  
element, see element variable  
global, see global variable  
nodal, see nodal variable  
variable names 19, 135, 137  
variable parameters 131, 133

## S

side set  
concatenated 15, 108, 112  
description 12  
distribution factors 14, 94, 96, 102, 104  
distribution factors,description 15  
element list 14  
ID 14  
node count list 15  
node list 14, 106  
node ordering 16  
number of 5, 29, 30, 31, 32  
parameters 14, 94, 96  
read 100  
read IDs 105, 128  
side list 14  
side numbering 13  
write 98

## T

testrd  
C version D-10  
Fortran version D-24

testwt  
C version D-1  
Fortran version D-17

time step  
description 19

time value  
description 19  
read 141, 143  
write 139

title 5, 29, 30, 31, 32  
truth table, see element variable

## U

update file 28

## V

version 5, 25, 26

## W

word size  
compute 23, 24, 25, 26  
I/O 23, 24, 25, 26  
I/O, description 5

## X

XDR 3