10/12/88 JSR

DRH0581-9

SANDIA REPORT

SAND87—2997 • UC—32
Unlimited Release
Printed September 1988

EXODUS: A Finite Element File Format for Pre- and Postprocessing

William C. Mills-Curran, Amy P. Gilkey, Dennis P. Flanagan

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

DISCLAIMER

SAND87-2997
Unlimited Release
Printed September 1988

Distribution
UC-32

EXODUS: A Finite Element File Format for Pre- and Postprocessing

SAND--87-2997

DE89 001054

William C. Mills-Curran
Amy P. Gilkey
Dennis P. Flanagan
Applied Mechanics Division III
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

The EXODUS format defines a binary file which is used for finite element analysis pre- and postprocessing. It includes data to define the finite element mesh and label both boundary condition and load application points. EXODUS accomodates multiple element types and is sufficiently general to service many different analysis codes. It also provides a very general format for analysis results. A benefit of combining the mesh definition data and the results data in the same file is that the user is assured that the results data are consistent with the model. EXODUS is currently in use by the entire range of Department 1520 codes (including preprocessors, translators, linear and nonlinear analyses, and postprocessors) and is finding applications in codes outside Department 1520.

3/4

Contents

Figures

Tables

1. INTRODUCTION

Ever since Department 1520 began to use pre- and postprocessing in conjunction with finite element analysis codes, the Department has sought to unify its computing environment wherever possible. That is, code specific pre- and postprocessing utilities have been avoided in favor of utilities which could service many analysis codes. For example, rather than writing a mesh generator which is tailored for a particular analysis code, the analysis code supporter works with the mesh generator supporter to insure that the two codes can communicate well.

The benefits of this approach are compelling:

- Each analysis code writer does not have to reinvent pre- and postprocessing independently. The analysis code writer then concentrates on what he/she does best – developing analysis code.
- Pre- and postprocessing efforts can be concentrated at a single point to maximize capabilities and support. This then reduces the total effort spent on pre- and postprocessing yet increases the probability that these capabilities will work correctly.
- Users of a new analysis code do not need to learn new pre- and postprocessors. Familiar pre- and postprocessors may be used for a wide range of analysis codes. Thus, a barrier to immediate use of a new analysis code is removed.

In order for this unified approach to succeed, a flexible communication format from preprocessor to analysis code to postprocessor must be defined. As Department 1520's finite analysis capabilities became more sophisticated, new requirements were placed on the communication link. In the past, the TAPE9, TAPE10, TAPE11 (SEACO), TAPE56, and GENESIS formats have provided this communication link. Each format represents a step forward in an evolutionary process as formats become both more sophisticated and more pliant.

Most recently, Department 1520 relied on the TAPE9 and SEACO formats as mesh definition and results data files for finite element analyses. A serious drawback of these formats is that they allowed only one type of element within the file. This has been acceptable for most nonlinear analyses within the department, because they typically use a single continuum element type. However, linear analyses frequently mix beams, shells, and solids within a single model and thus were restricted to the

PATRAN [1] format for pre- and postprocessing. (Special purpose translators are necessary to convert the data in a PATRAN file to a format which can be used by a finite element program.) In addition, there were significant differences between the TAPE9 (preprocessing or mesh) and the SEACO (postprocessing or results) binary files so that two different file formats were required to perform an analysis. The two different formats also needed their own translators to move data between different computers and operating systems.

As a first step to improving the situation, the GENESIS [2] format was defined. The GENESIS format implemented a multielement-type mesh format and successfully replaced the TAPE9 format.

The EXODUS format extends this benefit to results data by appending analysis results to the mesh definition. Thus, the beginning of an EXODUS file is a GENESIS file. In addition, the EXODUS format provides for quality assurance information to be carried through model creation, analysis, and results postprocessing steps.

In May, 1988, all applicable Department 1520 programs and procedures were successfully converted to the EXODUS format. It is expected that this standard will diffuse throughout the other departments in 1500 and become the de facto standard for finite element pre- and postprocessing files.

2. FILE DESCRIPTION

This chapter describes in detail the EXODUS format. It should be noted from the start that some of the records in an EXODUS file will appear only once, while others may appear multiple times. If a record contains no data, it still must be written. On systems (such as CTSS) which will not write a zero-length record, a single, arbitrary, datum should be written. When reading such a record, a null read is used.

An EXODUS file is read and written with standard FORTRAN unformatted input and output statements. Word packing is not used. No system dependent coding is required, thus, EXODUS will not complicate the movement of codes between computer systems.

An EXODUS file can logically be defined in two parts: A and B. Part A includes all data that can be defined by a preprocessor. These data are input to a finite element analysis code. The element type names record (Section 2.12) is the last record in part A. Part B contains the results of the analysis and begins with the results size record (Section 2.13). A postprocessor requires both parts to function properly.

Except for the first record of a time step, the information about the existence of a record and the length of a record is provided before the record is read. Thus, the format of a record is always known at the time it is read. Time step data are the only exception, because the EXODUS format uses an end-of-file to indicate the last record in the file.

Node and element numbers are indicated by the order in which the nodes and elements appear in the EXODUS file. Thus, the node and element numbers form a contiguous, ordered list.

In order to promote efficient storage within an EXODUS file and to enable efficient processing within analysis codes, elements are grouped into *element blocks*. Within an element block, all elements must be of the same type (basic configuration and number of nodes) and the same material.

In part B, analysis results are grouped into one of four different types for two different types of timesteps. *Element* results are output for all elements in one or more element blocks. Thus, element results can be described as a vector. Stress is an example of an element result. *Nodal* results are output for all the nodes in the model. Nodal results are also a vector. Displacement in the X direction is a nodal

result. *Global* and *history* results are output for a single element or node, or for a single property. Linear momentum of the structure and acceleration of node 5 are both examples of global and history variables.

History and global results are distinguished by the frequency at which they are output. Two different timesteps are defined in EXODUS. A *history timestep* contains only history results. A *whole timestep* contains history, global, nodal, and element results. No restriction is made on the frequency of either type of timestep, or that both appear in an EXODUS file. The only restriction is that time must increase monotonically. It can be seen that the history results and history timesteps provide a mechanism for chronologically refined output of select results.

Appendix A contains sample FORTRAN code which may be helpful in understanding the EXODUS format.

2.1 File Open

When opening an EXODUS file from FORTRAN, the only specific requirement is to include the `FORM='UNFORMATTED'` keyword and value in the OPEN statement.

2.2 Header Record

The first record of the EXODUS file is an 80-character string for use as a title or heading.

2.3 Problem Size Record

The problem size record consists of ten integers which define the general problem size and also identifies the version of the EXODUS format.

1. **Number of Nodes.** This datum defines the total number of nodes defined in the file.
2. **Problem Dimension.** The problem dimension refers to the number of space coordinates. A 1, 2, or 3 is required here.
3. **Number of Elements.** This is the total number of elements of all types in the file.
4. **Number of Element Blocks.** In EXODUS, elements are grouped together in blocks. Each block consists of elements of the same type and material. This datum defines the number of blocks.

5. **Number of Node Sets.** Node sets can be used as a convenient method for referring to groups of nodes. This datum defines the number of node sets in the file.

6. **Node Set List Length.** The node sets list length describes the total length of all node set lists. Because a node can appear in many node set lists, it is possible for the node set list length to be greater than the total number of nodes.

7. **Number of Side Sets.** Side sets are used to identify elements (and their sides) to which boundary conditions, such as heat flux, pressure, or slide lines, are applied. Elements in a single side set must be of similar type, e.g. all quads or all beams.

8. **Side Set Element List Length.** Each side set has a list of elements. This datum identifies the total length of all side set element lists. Because an element can appear in multiple side sets (or even repeat in a single side set), this list length can be greater than the total number of elements.

9. **Side Set Node List Length.** In order to identify the portion of an element to which a boundary condition is applied, the element's nodes must be included as part of a side set definition. This datum identifies the total length of all side set node lists.

10. **Version Number.** This is the EXODUS version number: 1. This version number allows future modifications to be made to EXODUS without causing undue confusion. The authors reserve the right to define/approve new versions of EXODUS.

2.4 Nodal Coordinates Record

This record contains the (real) spatial coordinates of all the nodes in the model. The number of nodes and the problem dimension (data 1 and 2 of the problem size record, Section 2.3) define the length of this record. The data cycle faster on nodes and slower on coordinates, thus the X coordinate for all the nodes is written before any Y coordinate data are written. Node numbers are implied from a node's place in the nodal coordinates record.

2.5 Optimized Element Order Map Record

This record defines the element order in which a wavefront solver should process the elements. For example, the first (integer) datum is the number of the element

which should be processed first by a wavefront solver. The length of this record is defined by datum 3 (number of elements) of the problem size record (Section 2.3). This record is necessary because the elements in EXODUS must be grouped in blocks and cannot be reordered for optimal wavefront solver performance.

When not used for indicating an optimized element order, this record can also be used to indicate a mapping to a different element order.

2.6 Element Blocks

For efficient storage and minimizing I/O, EXODUS groups elements in element blocks. Within an element block, all elements are of the same type (basic geometry and number of nodes) and have the same material definition.

The number of an element is defined by the order in which it appears in the EXODUS file. Elements are numbered consecutively across all element blocks.

The following three sections describe the three records which are repeated for each element block.

2.6.1 Element Block Parameter Record

This record defines four integer parameters which apply to this particular element block.

1. **Element Block ID.** The element block ID is an arbitrary, unique number which identifies the particular element block. This ID allows the user to specify a group of elements to the analysis or postprocessing code without having to know the order in which element blocks are stored in the file. For analysis, the user must associate each element block with an element type and a material model.

2. **Number of Elements.** This datum defines the number of elements in this element block.

3. **Nodes per Element.** This datum defines the number of nodes per element for this element type.

4. **Number of Attributes.** This datum defines the number of attributes per element.

2.6.2 Element Connectivity Record

This record defines the (integer) element connectivity for this element block. The length of this record is the product of data 2 and 3 (number of elements and nodes per element) of the element block parameters record. The node index cycles faster than the element index. Requirements for node ordering are found in Chapter 3.

2.6.3 Element Attributes Record

This record contains the (real) attributes for the elements in this element block, and its length is defined by data 2 and 4 (number of elements and attributes per element) of the element block parameters record. The attributes index cycles faster than the element index.

2.7 Node Sets

Node sets provide a means to reference a group of nodes with a single ID without requiring the user to know node numbers in the model. 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 set only once. Five records are used to define all the node set data in a model.

2.7.1 Node Set IDs Record

This record contains a list of unique integer node set IDs. The length of this record is defined by datum 5 (number of node sets) of the problem size record (Section 2.3).

2.7.2 Node Set Counts Record

Each integer datum in this record specifies the number of nodes in a particular node set list. While the i^{th} datum in the node set IDs record specifies a node set id, the i^{th} datum in the node set counts record indicates the number of nodes in that node set. The length of this record is defined by datum 5 (number of node sets) of the problem size record (Section 2.3).

2.7.3 Node Set Pointers Record

This record contains an integer pointer for each node set. Each pointer locates (in a concatenated list) the first node in the node set. The length of this record is defined by datum 5 (number of node sets) of the problem size record (Section 2.3).

2.7.4 Node Set Node List Record

This record is a concatenated (integer) list of all the nodes in node sets. The two previous records are used to find the first node and the number of nodes in a particular node set. The length of this record is defined by datum 6 (node set list length) of the problem size record (Section 2.3).

2.7.5 Node Set Distribution Factors

This record defines (real) distribution factors associated with the nodes in a node set. These data may be used for uneven application of load or boundary conditions. Each datum on this record is associated with the corresponding datum on the previous record (node set node list record) and is the same length.

2.8 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 elements in order to apply it properly. In order to retain simplicity in the definition of side sets, each side set can refer to only a single type of element. Within this restriction, however, a side set may contain elements from several element blocks.

Eight records are used to define side sets and have a format similar to that of the node set records.

2.8.1 Side Set IDs Record

This record contains a list of the unique integer side set IDs. The length of this record is defined by datum 7 (number of side sets) of the problem size record (Section 2.3).

2.8.2 Side Set Element Counts Record

This record specifies the integer number of elements in each side set and is the length defined by datum 7 (number of side sets) of the problem size record (Section 2.3).

2.8.3 Side Set Node Counts Record

This record specifies the integer number of nodes in each side set and is the length defined by datum 7 (number of side sets) of the problem size record (Section 2.3).

2.8.4 Side Set Element Pointer Record

This record contains an integer pointer for each element set which locates the first element of a particular side set within the side set element list. The length of this list is defined by datum 7 (number of side sets) of the problem size record (Section 2.3).

2.8.5 Side Set Node Pointer Record

This record contains an integer pointer for each element set which locates the first node in a particular side set within the side set node list. The length of this record is defined by datum 7 (number of side sets) of the problem size record (Section 2.3).

2.8.6 Side Set Element List Record

This record consists of a concatenation of the integer element lists for all side sets. The side set element pointer record indicates the location of the first element in a particular side set, and the side set element counts record indicates the number of elements in the side set. The length of this record is defined by datum 8 (side set element list length) of the problem size record (Section 2.3).

2.8.7 Side Set Node List Record

This record consists of a concatenation of the integer node lists for all side sets. The side set node pointer record indicates the location of the first node in a particular side set, and the side set node counts record indicates the number of nodes in the side set. The length of this record is defined by datum 9 (side set node list length)

of the problem size record (Section 2.3). Requirements for node ordering are found in Chapter 3.

2.8.8 Side Set Distribution Factors Record

This record contains the distribution factors associated with the nodes in a side set. This data may be used for uneven application of load or boundary conditions. Each real datum on this record is associated with the corresponding datum on the previous record and is the same length.

2.9 QA Data

Quality Assurance (QA) data are contained on two or more records.

2.9.1 QA Count Record

This record contains a single integer datum which indicates the number of QA title records which follow. Each file must contain at least one QA title record.

2.9.2 QA Title Records

Each QA title record consists of four eight-character strings. The number of QA title records is indicated by the QA count record and must be at least one. In the case where multiple codes modify the file, each code should copy all existing QA records and append an additional QA record.

1. **Code Name.** The code name indicates the code which has operated on the EXODUS file.

2. **Code QA descriptor.** The code QA descriptor provides a location for a version identifier.

3. **Date.** The date field should be of the format `01/25/88`.

4. **Time.** The time datum indicates the 24 hour time with fields for hours, minutes and seconds: `16:30:15`.

2.10 Information Data

The following records provide a location for storage of supplementary text. There is no minimum number of text records, but the count record is required.

2.10.1 Information Count Record

This record contains a single integer datum which indicates the number of information text records to follow.

2.10.2 Information Text Records

This is an 80-character text record which is intended for optional supporting documentation from a code. Any number of records may be used and the number is indicated by the information count record.

2.11 Coordinate Names Record

This record contains eight-character fields which name the spatial coordinates. There is one field for each dimension in the model (see datum 2 of the problem size record, Section 2.3), thus there are one to three fields on the record. Coordinate naming conventions are in Section 3.4.

2.12 Element Type Names Record

Element type names are included to uniquely distinguish element types. This record contains an 8-character field for each element block. Names for several element types are listed in Section 3.3.

2.13 Results Size Record

The results size record contains four integer data.

1. **Number of History Variables**
2. **Number of Global Variables**
3. **Number of Nodal Variables**
4. **Number of Element Variables**

2.14 Results Names Record

This record contains the character string names (8 characters each) which are associated with the history, global, nodal, and element results. The names are organized in four groups, and the size of each group is defined on the results size record. Any group size (including zero) is permitted.

1. **History Variable Names**
2. **Global Variable Names**
3. **Nodal Variable Names**
4. **Element Variable Names**

2.15 Truth Table Record

The integer data in the truth table record indicates 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 in element block 6.

It is helpful to describe the truth table as a two dimensional array. Each row of the array is associated with an element variable and element variable name. Each column of the array is associated with an element block. If a datum in the truth table is 0, then no results are output for that element variable for that element block. A nonzero entry indicates that the appropriate results will be output.

The truth table array is stored in columns on the record, that is, the variable index cycles faster than the block index.

2.16 Time Step Data

The number of records associated with time step data is quite variable for two reasons. First, no provision is made to record the number of time steps in the file because analysis codes frequently stop (or fail) at unpredictable times. Second, history and whole timesteps contain different numbers of records. (See the discussion at the beginning of this chapter.) History timesteps include only two records, while whole timesteps contain additional records with global, nodal, and element results.

2.16.1 Time Record

This record contains two real data: the analysis time for the results that follow in the file, and a flag which indicates whether this is a history timestep or a whole timestep. The value 0.0 indicates that this is a whole timestep. Any other value indicates a history timestep.

2.16.2 History Results Record

The history results record contains the (real) history data and is present for history and whole timesteps. The length of this record is indicated by datum 1 (number of history variables) of the results size record (Section 2.13).

2.16.3 Global Results Record

This record contains the (real) global data and is present only for whole timesteps. The length of this record is indicated by datum 2 (number of global variables) of the results size record (Section 2.13).

2.16.4 Nodal Results Records

These records contain the (real) nodal data and are present only for whole timesteps. There is a record for each nodal variable name, thus the number of records is indicated by datum 3 (number of nodal variables) of the results size record (Section 2.13). The length of these records is indicated by datum 1 (number of nodes) of the problem size record (Section 2.3).

2.16.5 Element Results Records

Element results records are written from within nested loops. The outer loop cycles on the element blocks, and the inner loop cycles on the element variables. An element results record is output for a particular combination of element block and element variable only if indicated by the truth table (Section 2.15). If output is indicated, then the record contains one datum for each element in one element block.

3. CONVENTIONS

This chapter contains supplementary requirements for use of the EXODUS file format. The preceding chapter provides a complete description of the *structure* of the file, but is incomplete as far as content and usage are concerned. For example, provisions are made for storing element type names. These names are necessary to uniquely identify some types of elements (e.g. tetrahedron vs. quadrilateral). But the file format does not suggest which specific names should be used (e.g. TET vs. TETRA or QUAD vs. QUAD4). Thus, it becomes necessary to impose conventions for use of the EXODUS format.

These conventions are provided to smooth the use of the EXODUS format definition. Department 1520 users of the EXODUS format have agreed to adhere to these conventions, thus, any EXODUS data which does not follow these conventions may not interface with Department 1520 codes.

3.1 Element Connectivity Node Ordering

The element connectivity should be ordered in the same manner as PATRAN [1]. Figure 3.1 depicts the node ordering for six higher-ordered elements. Lower-ordered elements should use the same ordering scheme, skipping the nodes that are not used in the element.

3.2 Side Set Node Ordering

Nodes in a side set should be ordered to help the analysis code identify the "outward" direction. Figure 3.2 shows the node ordering for higher ordered elements. For lower ordered elements, simply omit the unused nodes.

3.3 Element Names and Attributes

An element naming convention is required for correct plotting of elements. Element attributes are necessary to completely describe structural (non-continuum) elements. Table 3.1 lists the names of several types of elements and the attributes that are normally expected for each.

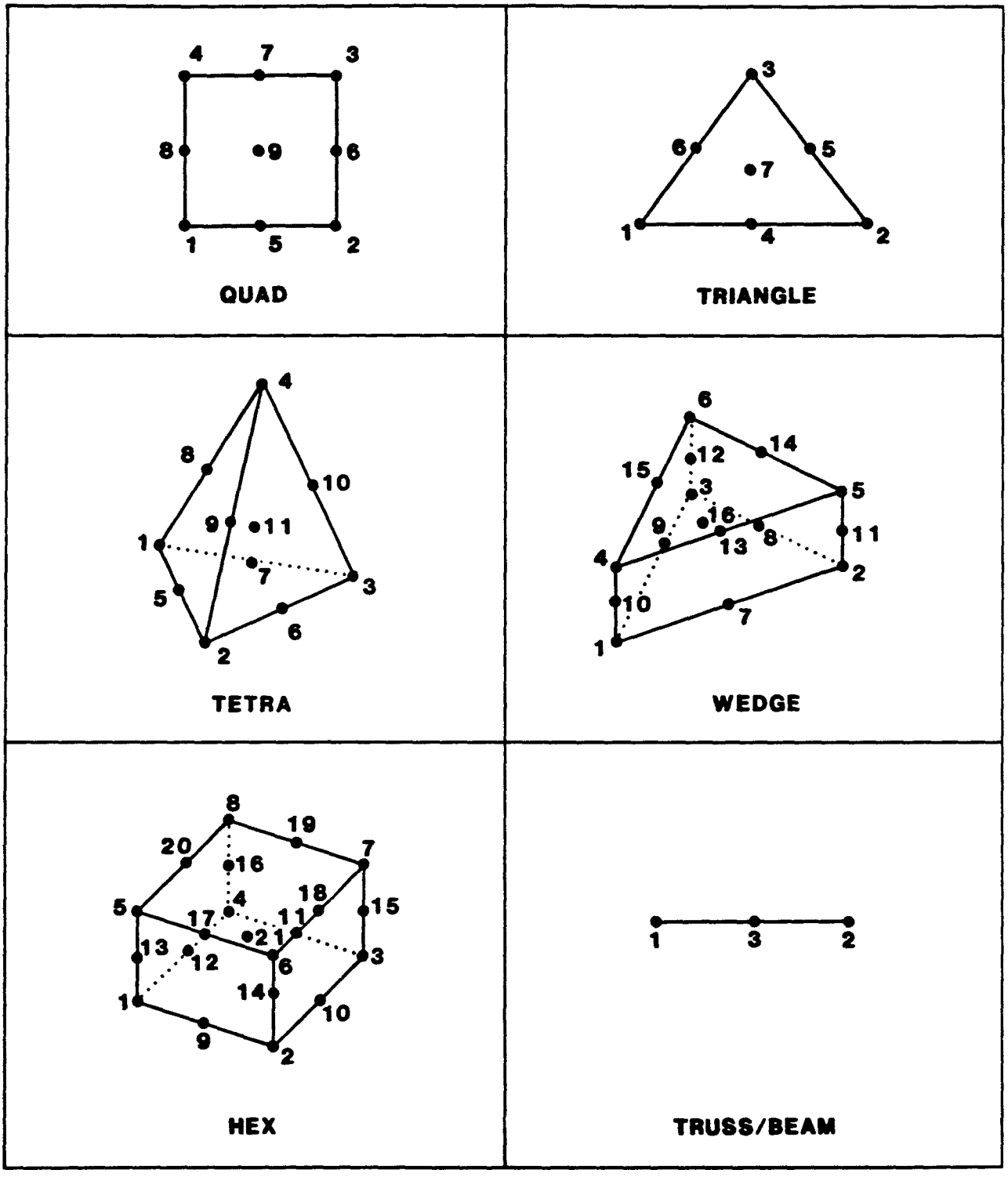

Figure 3.1. Element Node Order

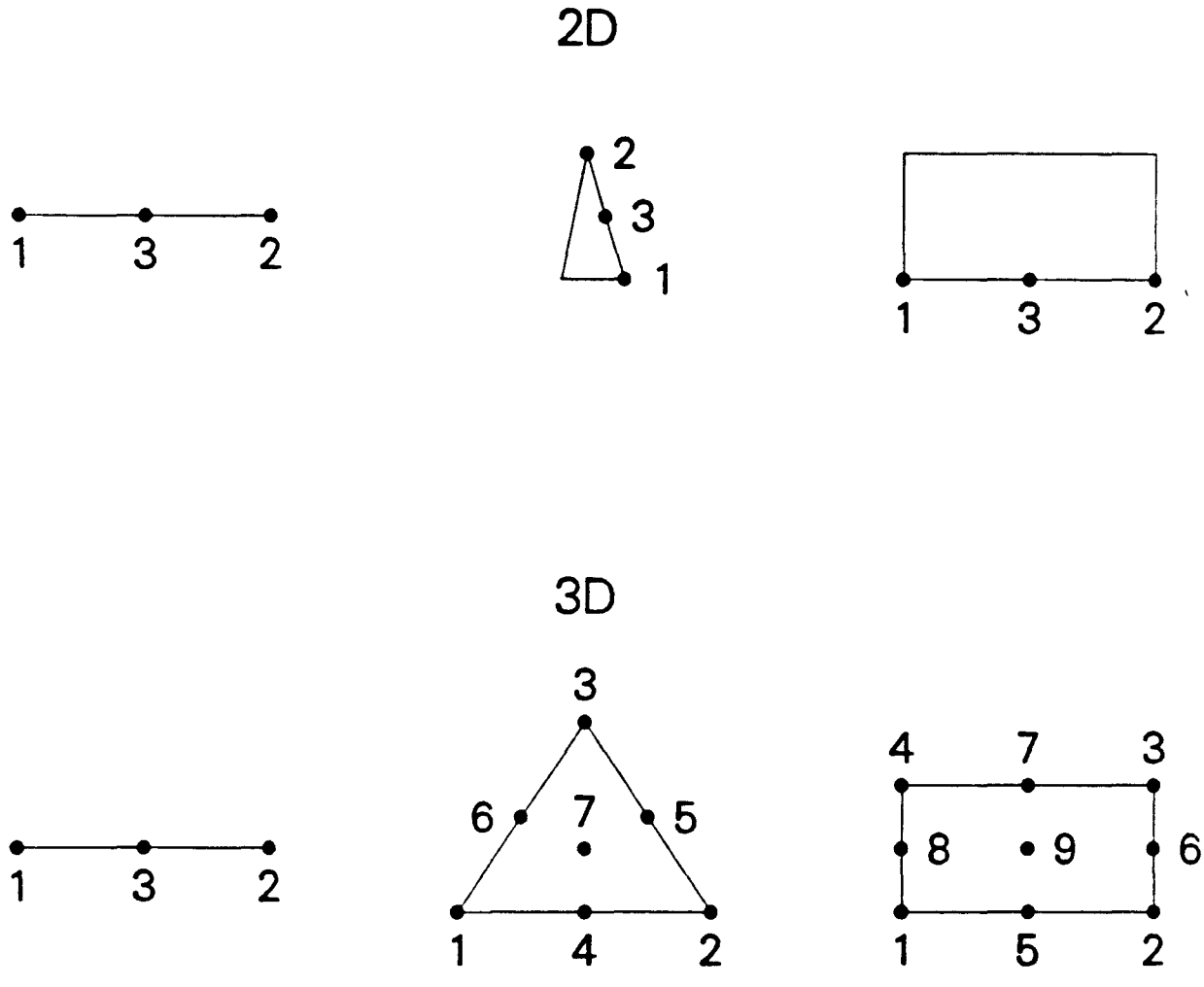

Figure 3.2. Side Set Node Ordering

3.4 Coordinate and Displacement Results Names

In order for postprocessing codes to link the proper coordinates with displacement results, the following rules should be followed:

1. The displacement names must be the first 3 (or 2 for 2D) names in the nodal variable list.
2. All displacement names must begin with "D" and must be the same except for the last character.
3. The last character of the i^{th} coordinate name must match the last character of i^{th} displacement name.

3.5 Character Strings

All character strings should be left justified in their field.

Table 3.1. Element Type Names and Attributes

NAME	ATTRIBUTES
TRUSS	Area
BEAM	3D: Area, I_1, I_2, J, V_1, V_2, V_3
	2D: Area, I, J
TRIANGLE	t
QUAD	t
TETRA	
WEDGE	
HEX	
CIRCLE	r
SPHERE	r

The V_i define a vector that, together with the axis of the element defines a plane for the beam element. The I_1 bending moment of inertia affects displacements in this plane, while the I_2 bending moment of inertia affects bending out of this plane. J is the torsional (polar) moment of inertia.

A. FORTRAN PROGRAMMING EXAMPLE

The program fragment below shows how an EXODUS file may be written. Because this is a program fragment, many of the statements necessary to make this a legal program are missing, but sufficient detail is included to aid the reader in understanding the EXODUS format.

```
      CHARACTER HEAD*80
C
C   --Open the database file
C
      NDB = 11
      OPEN (UNIT=NDB, ..., FORM='UNFORMATTED')
C
C   --Header record
C
      WRITE (NDB) HEAD
C
C   --Problem size record
C
      WRITE (NDB) NUMNOD, NDIM, NUMEL, NELBLK,
     *          NUMNPS, LNPSNL, NUMESS, LESSEL, LESSNL, 1
C       --NUMNOD - the number of nodes
C       --NDIM - the number of dimensions
C       --NUMEL - the total number of elements
C       --NELBLK - the number of element blocks
C       --NUMNPS - the number of node sets
C       --LNPSNL - node set list length
C       --NUMESS - number of side sets
C       --LESSEL - side set element list length
C       --LESSNL - side set node list length
C       --1 - version number of this EXODUS format
C
      WRITE (NDB) ((COORD(I,J),I=1,NUMNOD),J=1,NDIM)
C       --COORD - Nodal Point Coordinates
C
      WRITE (NDB) (MAP(I),I=1,NUMEL)
C       --MAP - Optimized Element Order Map
```

```
C
C   --Element Blocks
C
      DO 100 K=1,NELBLK
C
C      --Sizing Parameters for this Element Block
C
        WRITE (NDB) IDEBLK,NUMELB,NELNOD,NATRIB
C         --IDEBLK - Element block identification (must be unique)
C         --NUMELB - Number of elements in this block (the sum of
C                    NUMELB over all blocks must equal NUMEL above)
C         --NELNOD - Number of nodes defining the connectivity for an
C                    element of this type
C         --NATRIB - Number of element attributes
C                    for this element type
C
        WRITE (NDB) ((ICONK(J,I),J=1,NELNOD),I=1,NUMELB)
C         --ICONK - Connectivity for this Element Block
C
        WRITE (NDB) ((ATRIBK(J,I),J=1,NATRIB),I=1,NUMELB)
C         --ATRIBK - Attributes for this Element Block
C
  100 CONTINUE
C
C   --Node Sets Data
C
      WRITE (NDB) (IDNPS(I),I=1,NUMNPS)
C      --IDNPS - Node Set IDs
C
      WRITE (NDB) (NNNPS(I),I=1,NUMNPS)
C      --NNNPS - Node Set Counts
C
      WRITE (NDB) (IPTNPS(I),I=1,NUMNPS)
C      --IPTNPS - Node Set Pointers
C
      WRITE (NDB) (LSTNPS(I),I=1,LNPSNL)
C      --LSTNPS - Node Set Node List
C
      WRITE (NDB) (FACNPS(I),I=1,LNPSNL)
C      --FACNPS - Node Set Distribution Factors
C
```

```
C   --Side Sets Data
C
      WRITE (NDB) (IDESS(I),I=1,NUMESS)
C      --IDESS - Side Set IDs
C
      WRITE (NDB) (NEESS(I),I=1,NUMESS)
C      --NEESS - Side Set Element Counts
C
      WRITE (NDB) (NNESS(I),I=1,NUMESS)
C      --NNESS - Side Set Node Counts
C
      WRITE (NDB) (IPEESS(I),I=1,NUMESS)
C      --IPEESS - Side Set Element Pointers
C
      WRITE (NDB) (IPNESS(I),I=1,NUMESS)
C      --IPNESS - Side Set Node Pointers
C
      WRITE (NDB) (LTEESS(I),I=1,LESSEL)
C      --LTEESS - Side Set Element List
C
      WRITE (NDB) (LTNESS(I),I=1,LESSNL)
C      --LTNESS - Side Set Node List
C
      WRITE (NDB) (FACESS(I),I=1,LESSNL)
C      --FACESS - Side Set Distribution Factors
C
C
C   --Write the QA header information
C
      WRITE (NDB) NQAREC
C      --NQAREC - the number of QA records (must be at least 1)
C
      DO 110 IQA = 1, NQAREC
         WRITE (NDB) (QATITL(I,IQA), I=1,4)
C         --QATITL - the QA title records; each record contains:
C         --    1) analysis code name (CHARACTER*8)
C         --    2) analysis code qa descriptor (CHARACTER*8)
C         --    3) analysis date (CHARACTER*8)
C         --    4) analysis time (CHARACTER*8)
  110 CONTINUE
C
```

```
C   --Write the optional header text
C
      WRITE (NDB) NINFO
C      --NINFO - the number of information records
C
      DO 120 I = 1, NINFO
         WRITE (NDB) INFO(I)
C         --INFO - extra information records (optional) that
C         --   contain any supportive documentation that the
C         --   analysis code developer wishes (CHARACTER*80)
  120 CONTINUE
C
C   --Write the coordinate names
C
      WRITE (NDB) (NAMECO(I), I=1,NDIM)
C      --NAMECO - the coordinate names (CHARACTER*8)
C
C   --Write the element type names
C
      WRITE (NDB) (NAMELB(I), I=1,NELBLK)
C      --NAMELB - the element type names (CHARACTER*8)
C
C     END OF EXODUS PART A - BEGINNING OF PART B
C
C   --Write the history, global, nodal, and
C     element variable information
C
      WRITE (NDB) NVARHI, NVARGL, NVARNP, NVAREL
C      --NVARHI - the number of history variable names
C      --NVARGL - the number of global variable names
C      --NVARNP - the number of nodal variable names
C      --NVAREL - the number of element variable names
C
      WRITE (NDB)
     &   (NAMEHI(I), I=1,NVARHI),
     &   (NAMEGV(I), I=1,NVARGL),
     &   (NAMENV(I), I=1,NVARNP),
     &   (NAMEEV(I), I=1,NVAREL)
C      --NAMEHI - the history variable names (CHARACTER*8)
C      --NAMEGV - the global variable names (CHARACTER*8)
C      --NAMENV - the nodal variable names (CHARACTER*8)
```

```
C      --NAMEEV - the element variable names (CHARACTER*8)
C
       WRITE (NDB) ((ISEVOK(I,J), I=1,NVAREL), J=1,NELBLK)
C      --ISEVOK - the name truth table for the element blocks;
C      --   ISEVOK(i,j) refers to variable i of element block j;
C      --   the value is 0 if and only if data will NOT be output for
C      --   variable i for element block j (otherwise the value is 1)
C
C*******************************************************
C
C   --Write each time step
C     (the time value must increase monotonically)
C
      DO 160 ITIME = 1, NSTEP
C
         WRITE (NDB) TIME(ITIME), HIFLAG(ITIME)
C         --HIFLAG - the history flag (0 for output of all variables
C                                     nonzero for history variables only)
C         --TIME - the time step value
C
         WRITE (NDB) (VALHI(IVAR,ITIME), IVAR=1,NVARHI)
C         --VALHI - the history values for the time step
C
         IF (HIFLAG(ITIME) .EQ. 0.) THEN
C
            WRITE (NDB) (VALGV(IVAR,ITIME), IVAR=1,NVARGL)
C            --VALGV - the global values for the time step
C
            DO 130 IVAR = 1, NVARNP
               WRITE (NDB) (VALNV(INP,IVAR,ITIME),
     &            INP=1,NUMNOD)
C               --VALNV - the nodal variables at each node
C               --   for the current time step
  130       CONTINUE
C
            DO 150 IBLK = 1, NELBLK
               DO 140 IVAR = 1, NVAREL
                  IF (ISEVOK(IVAR,IBLK) .NE. 0) THEN
                     WRITE (NDB) (VALEV(IEL,IVAR,IBLK,ITIME),
     &                  IEL=1,NUMELB(IBLK))
C                     --VALEV - the element variables at each element
```

```
C                     --   for the current time step
                   END IF
  140            CONTINUE
  150          CONTINUE
C
          END IF
  160 CONTINUE
C
C*********************************************************
C
C   --End of data is indicated by an end of file
C
       CLOSE (NDB)
```

References

[1] "PATRAN Plus User Manual," PDA Engineering, Costa Mesa, CA, 1987.

[2] L. M. Taylor, D. P. Flanagan, and W. C. Mills-Curran, "The GENESIS Finite Element Mesh File Format," Sandia National Laboratories, Albuquerque, NM, SAND86-0910, May 1986.

DISTRIBUTION

1510 J. W. Nunziato
1511 N. E. Bixler
1511 R. R. Eaton
1511 D. K. Gartling
1511 R. C. Givler
1511 J. H. Glick
1511 P. L. Hopkins
1512 J. C. Cummings, Jr.
1513 D. W. Larson
1513 J. A. Schutt
1513 C. E. Sisson
1520 L. W. Davison
1521 R. D. Krieg
1521 J. D. Miller
1521 G. D. Sjaardema
1521 C. M. Stone
1521 J. R. Weatherby
1522 R. C. Reuter, Jr.
1522 C. R. Adams
1522 T. D. Blacker
1522 R. J. Kipp
1522 D. R. Martinez
1523 J. H. Biffle
1523 D. P. Flanagan (20)
1523 A. P. Gilkey
1523 J. R. Koteras
1523 W. C. Mills-Curran (20)
1523 L. M. Taylor
1524 A. K. Miller
1524 K. W. Gwinn
1524 J. Pott
1530 W. Herrmann (actg.)
1550 C. W. Peterson
2814 P. F. Chavez
3141 S. A. Landenberger (5)
3141-2 for DOE/OSTI (8)
3151 W. I. Klein (3)
6416 R. P. Rechard
8231 M. H. Pendley
8231 V. K. Gabrielson
8240 C. W. Robinson
8241 G. A. Benedetti
8242 M. R. Birnbaum
8243 M. L. Callabresi
8243 W. E. Mason, Jr.
8244 C. M. Hartwig
8245 R. J. Kee, Jr.
8524 P. W. Dean