Tecplot
Enjoy the View

TECPLOT 360 2006

Data Format Guide

Tecplot, Inc.   **Bellevue, WA**   **2006**

Chapter 1 # Best Practices

Tecplot can read in data produced in many different formats, one of which is its own native format. Users who wish to generate native Tecplot data files automatically from applications such as complex flow solvers have a number of options for what to put in the file and how it is made. This section outlines a few "best practices" for writing Tecplot data files.

1 - 1 Create Binary Data Files instead of ASCII

All else being equal, binary data files are more efficient than ASCII files, in terms of space and time. To create binary data files, you must make calls to functions provided by the TECIO library (See the 3 - 7 "Binary Data File Function Reference" on page 74). To create ASCII files, you can write-out plain text using standard write statements.

There are some cases where ASCII files are preferred. Create ASCII files when:

- Your data files are small.

- Your application runs on a platform for which the TECIO library is not provided. Even if this is the case, please contact Tecplot Inc. There may be a way to resolve this issue.

> Tecplot includes a utility called Preplot which allows you to convert an ASCII file into a binary file. See "Preplot" in the Tecplot 360 User's Manual for more information on how to use Preplot.

1 - 2 Use Block Format instead of Point Format

Block format is by far the most efficient format when it comes to loading the file into Tecplot. If your data files are small and you can only obtain the data in a point-like format in (like a spreadsheet), then using point format is acceptable.

> NOTE: ASCII files in point format will be in point format when converted to binary format using Preplot.

1 - 3 Use the Native Byte Ordering for the Target Machine

When you create binary data, you can elect to produce these files in either Motorola byte order or Intel byte order. Tecplot automatically detects the byte order and loads both types. However, it is more efficient if you produce files using the byte order used on the platform where you run Tecplot. For example if you produce a binary file on an SGI platform and then transfer the data to a Windows or Intel-based Linux box, you should to set the flag to reverse the bytes when generating the binary data file. See the notes about this option in "Preplot" in the Tecplot 360 User's Manual for the Preplot flag.

1 - 4 Add Auxiliary data to Preset Variable Assignments in Tecplot

Zone Auxiliary data can be used to give Tecplot hints about properties of your data. For example, this can be used to like set the defaults for which variables to use for certain kinds of plots. Auxiliary data is supported by both binary and ASCII formats. For a list of auxiliary data names, see "Using Standardized Auxiliary Data" in the ADK Users manual.

1 - 5 Data Sharing

Share variables whenever possible. Variable sharing is commonly used for the spatial variables (X, Y, and Z) when you have many sets of data that use the same basic grid. This saves both disk space and space when loaded into Tecplot. In addition, the benefits are compounded with scratch data derived from these variables because it is also shared within Tecplot. See also Data Sharing in the Tecplot 360 User's Manual.

1 - 6 Passive Variables

Tecplot can manage many datasets at the same time. However, within a given dataset you must supply the same number of variables for each zone. In some cases you may have data where there are many variables and, for some of the zones some of those variables are not important. If that is the

case you can make selected variables in selected zones passive. A passive variable is one that will always return the value zero if queried (like in a probe) but will not involve itself in operations like the calculations of the min and max range. This is very useful when it comes to calculating such as default contour levels.

Chapter 2 **ASCII Data**

This chapter discusses how to format data so data files may be loaded directly into Tecplot. Data files read by Tecplot may be binary or ASCII. The following sections describe the format of ASCII data files. Reading an ASCII data file into Tecplot can be much slower than reading a binary data file, as binary data files take up less disk space and Tecplot must convert from ASCII to binary.

Tecplot or Preplot converts ASCII data files to binary. See Tecplot Format Data File Loading in the Tecplot 360 User's Manual for converting with Tecplot, or Section 2- 4.2, "ASCII Data File Conversion to Binary," for converting with Preplot. Documentation on the binary format is included as comments in the Preplot source code (located in TEC360HOME/Util/preplot). Finally, if your data is generated in FORTRAN or C, you may be able to generate binary data files directly using the utilities described in 3, "Binary Data,".

You can also load data generated by, or tabulated in, other software packages. In addition, data loaders using Tecplot's Add-on Developer's Kit (ADK) are available from Tecplot. These convert data from a number of software packages into a Tecplot-readable format. (This is described in Data Loaders in the Tecplot 360 User's Manual) Using ADK, you can write loaders of your own.

2 - 1 ASCII Data File Records

An ASCII data file begins with an optional file header defining a title for the data file and or the names of the variables. The header is followed by optional zone records containing the plot data. Zone records may contain ordered or finite-element data. You may also include text, geometry, and custom-label records that create text, geometries, and/or custom labels on plots. Each data file may have up to 32,700 zone records, ten custom label records, and any number of text and geometry records. These records may be in any order.

The first line in a zone, text, geometry, custom label, or data set auxiliary data record begins with the keyword **ZONE**, **TEXT**, **GEOMETRY**, **CUSTOMLABELS**, **DATASETAUXDATA, or VARAUX-DATA.** The maximum length of a line in a data file is 4,000 characters (unless you edit and recompile the Preplot source code). Any line may be continued onto one or more following lines (except for text enclosed in double quotes [**"**]). Double quotes must be used to enclose character strings with embedded blank spaces or other special characters. A backslash (****) may be used to remove the significance of (or escape) the next character (that is, **\"** produces a single double-quote). Any line beginning with an octothorp (**#**) is treated as a comment and ignored.

The following simple example of a Tecplot ASCII data file has one small zone and a single line of text:

```
TITLE="Simple Data File"
VARIABLES="X" "Y"
ZONE I=4 F=POINT
1 1
2 1
2 2
1 2
TEXT X=10 Y=90 T="Simple Text"
```

The format of the ASCII data file is summarized in Section , "Summary of Data File Records".

File Header

In the file header of your data file, you may specify an optional title that is displayed in the headers of Tecplot frames. The title line begins with **TITLE=**, followed by the title text enclosed in double-quotes. You may also assign a name to each of the variables by including a line that begins with **VARIABLES=**, followed by each variable's name enclosed in double quotes. The quoted variable names should be separated by spaces or commas. Tecplot calculates the number of variables (*N*) from the list of variable names. If you do not specify the variable names (and your first zone has **POINT** data packing), Tecplot sets the number of variables equal to the number of numeric values in the first line of zone data for the first zone, and names the variables **V1**, **V2**, **V3**, and so forth.

Initially, Tecplot uses the first two variables in data files as the X- and Y-coordinates, and the third variable for the Z-coordinate of 3D plots. You may, however, order the variables in the data file any way you want, since you can interactively reassign the variables to the X-, Y-, and or Z-axes using Tecplot dialogs.

Dataset and variable auxiliary data is added to the datafile using the DATASETAUXDATA and VARAUXDATA records. Auxiliary data are name/value pairs that a user can specify and then use in Tecplot with dynamic text, equations, macros, or add-ons. Multiple auxiliary data can be added at the dataset level as follows:

```
DATASETAUXDATA SampleNumber="5"
DATASETAUXDATA AOA="5.7"
```

Variable auxiliary data is added to Tecplot on a per variable basis. Like dataset auxiliary data multiple items can be added for each variable:

```
VARAUXDATA 1 MyData="Hello"
VARAUXDATA 1 MoreData="World"
VARAUXDATA 2 MyData="More information"
VARAUXDATA 2 MoreData="hi mom"
VARAUXDATA 2 MyExtraData="Some extra data"
```

The variable number with which the auxiliary data is associated immediately follows the VARAUXDATA record. Also note that the data associated with a particular auxiliary data name are unique for each variable. Therefore the same named item can be added to each variable if desired. Conversely a particular auxiliary data item can be added to only one variable.

If the file header occurs in a place other than at the top of the data file, a warning is printed and the header is ignored. This allows you to concatenate two or more ASCII data files before using Tecplot (provided each data file has the same number of variables per data point).

Zone Records

A zone record consists of a control line that begins with the keyword **ZONE** followed by a set of numerical data called the zone data. The format of the zone control line is shown in Section , "Summary of Data File Records".

The ZONETYPE Parameter. The zone data are of the type specified by the **ZONETYPE** parameter in the control line. There are two basic types of zones: ordered and finite-element. Ordered zones have the formats **ZONETYPE=ORDERED**. Finite-element zones have the specific **ZONETYPE** of **FELINESEG**, **FETRIANGLE**, **FEQUADRILATERAL**, **FETETRAHEDRON**, or **FEBRICK**. **ORDERED** is presumed if the **ZONETYPE** parameter is omitted. See Section 2 - 3, "Ordered Data,"for more information on ordered zones, and Section 2 - 4, "Finite-Element Data,"for details on finite-element data.

The DATAPACKING Parameter. The zone data packing is specified by the **DATAPACKING** parameter in the control line. There are two data packing options: **POINT** and **BLOCK**.

In **POINT** format, the values for all variables are given for the first point, then the second point, and so on. In **BLOCK** format, all of the values for the first variable are given in a block, then all of the values for the second variable, then all of the values for the third, and so forth. Zones with cell-centered data must use **BLOCK** data packing. More detail on this is given below.

POINT Format Example. If you have only one zone of data in **POINT** data packing format, and it is one-dimensional (that is, *JMax=1*, *KMax =1*), you may omit the zone control line. If you want Tecplot to determine the number of variables, you may create a data file with only the zone data, such as the following:

```
12.5 23 45 1.
14.3 24 46 2.
12.2 24 50 3.
13.3 26 51 4.
13.5 27 55 5.
```

Tecplot calculates the number of data points (*IMax*) in the zone by assuming that each row represents a data point and each column represents a variable, and creates an I-ordered zone. This type of structure is good for XY-plots and scatter plots. If there are multiple zones, two- or three-dimensional zones, finite-element zones, or **BLOCK**-format zone data, you must include a zone control line at the beginning of each zone record.

Data Types. Each variable in each zone in the data file may have its own data type. Tecplot supports the following six data types:

- **DOUBLE** (eight-byte floating point values).

- **SINGLE** (four-byte floating point values).

- **LONGINT** (four-byte integer values).

- **SHORTINT** (two-byte integer values).

- **BYTE** (one-byte integer values, from 0 to 255).

- **BIT**

The data type determines the amount of storage Tecplot assigns to each variable. Therefore, the lowest level data type should be used whenever possible. For example, imaging data, which usually consists of numerical values ranging from zero to 255, should be given a data type of **BYTE**. By default, Tecplot treats numeric data as data type **SINGLE**. If any variable in the zone uses the **BIT** data type, the zone format must be **BLOCK** or **FEBLOCK**; you cannot use **POINT** or **FEPOINT** format.

Variable Location. Each variable in each zone in a data file may be located at the nodes or the cell-centers. Each variable is specified as **NODAL** or **CELLCENTERED** in the **VARLOCATION** parameter array, located in the control line. The format is:

   VARLOCATION=([*set-of-vars***]=var-location,[***set-of-vars***]=var-location, ...)**

where *set-of-vars* is the set of the variables and var-location is either **NODAL** or **CELLCENTERED**. Variables omitted from the list are assumed to be **NODAL**. For example:

   VARLOCATION=([3-7,10]=CELLCENTERED, [11-12]=CELLCENTERED)

specifies that variables 3 through 7, 10, 11 and 12 are cell-centered and all other variables are, by default, nodal for this zone.

All cell-centered variables must list one value for each element. With nodal variables, one value must be listed for each node. Zones with cell-centered variables must be in **BLOCK** data packing format.

Data Lists. Numerical values in zone data must be separated by one or more spaces, commas, tabs, new lines, or carriage returns. Blank lines are ignored. Integer (**101325**), floating point (**101325.0**), and exponential (**1.01325E+05**) numbers are accepted. To repeat a particular number in the data, precede it with a repetition number as follows: "*Rep*Num*," where *Rep* is the repetition factor and *Num* is some numeric value to be repeated. For example, you may represent 37 values of 120.5 followed by 100 values of 0.0 as follows:

```
37*120.5, 100*0.0
```

Zone Auxiliary Data. Auxiliary data strings associated with the current zone are specified with **AUXDATA** parameter in the control line. This auxiliary data may be used in dynamic text, equations, macros, or add-ons. Auxiliary data is provided as named strings:

```
AUXDATA EXPERIMENTDATE ="October 13, 2002, 8 A.M."
```

There may be multiple **AUXDATA** parameters in the control line for a zone, but names must be unique.

Variable Sharing between Zones. Frequently, some variables are exactly the same for a set of zones. For example, a series of zones may contain measurement or simulation data at the same XYZ-locations, but different times. In this case, Tecplot's memory usage may be dramatically reduced by sharing the coordinate variables between the zones. The zones that variables are shared from are specified in the **VARSHARELIST** in the control line of the current zone. The format is:

```
VARSHARELIST=([set-of-vars]=zzz, [set-of-vars]=zzz)
```

where *set-of-vars* is the set of variables that are shared and *zzz* is the zone they are shared from. If *zzz* is omitted, the variables are shared from the previous zone. For example:

```
VARSHARELIST=([4-6,11]=3, [20-23]=1, [13,15])
```

specifies that variables 4, 5, 6 and 11 are shared from zone 3, variables 20, 21, 22, and 23 are shared from zone 1, and variables 13 and 15 are shared from the previous zone. For variable sharing, ordered zones may only share with ordered zones having the same dimensions. Finite-element zones may share with any zone having the same number of nodes, for nodal variables, or the same number of cells, for cell-centered data.

The connectivity list (finite-element only) and face-neighbors may be shared between zones using the **CONNECTIVITYSHAREZONE** parameter in the control line of the current zone. The format is:

CONNECTIVITYSHAREZONE=*nnn*

where *nnn* is the number of the zone that the connectivity is shared from. To use connectivity sharing, the zone must have the same number of points and elements, and be the same zone type.

Face Neighbors. The implicit connections between elements in a zone may be overridden, or connections between cells in adjacent zones established, using the **FACENEIGHBORMODE** parameter and **FACENEIGHBORCONNECTIONS** list in the control line of the zone. **FACENEIGHBORMODE** has four options: **LOCALONETOONE**, **LOCALONETOMANY**, **GLOBALONETOONE**, and **GLOBALONETOMANY**. **LOCALONETOONE** is the default.

The nature of the **FACENEIGHBORCONNECTIONS** list depends upon the **FACENEIGHBORMODE**, described in the table below. To connect the cells along one edge to cells on another edge of the same zone, use **LOCAL**. To connect cells of one zone to cells of another zone or zones, use **GLOBAL**. If the points of the cells are exactly aligned with the neighboring cell points, use **ONETOONE**. If even one cell face is neighbor to two other cell faces, use **ONETOMANY**.

Mode	Number of Values	Data
LOCALONETOONE	`3`	**cz, fz, nc**
LOCALONETOMANY	`nz+4`	**cz, fz, oz, nz, nc1, nc2, ..., ncn**
GLOBALONETOONE	`4`	**cz, fz, zr, cr**
GLOBALONETOMANY	`2*nz+4`	**cz, fz, oz, nz, zrl, crl, zr2, cr2, ..., zrn, crn**

In this table, **cz** is the cell number in the current zone, **fz** is the number of the cell face in the current zone, **nc**, is the cell number of the neighbor cell in the current zone, **oz** is face obscuration flag (zero for face partially obscured, one for face entirely obscured), **nz** is the number of neighboring cells for the **ONETOMANY** options, **ncn** is the number of the *nth* local zone neighboring cell in the list, **zr** is the remote zone number, **cr** is the cell number of the neighboring cell in the remote zone, **zrn** is the zone number of the nth neighboring cell in the **GLOBALONETOMANY** list, and **crn** is the cell number in the remote zone of the *nth* neighboring cell in the **GLOBALONETOMANY** list.

The **cz**, **fz** combinations must be unique; multiple entries are not allowed. The face numbers for cells in the various zone types are defined in Figure 2-1.

Figure 2-1. Examples of brick (left) and tetrahedron (right) face neighbors.

A connection must be specified for two matching cell faces to be effective. For example, for data with a FACENEIGHBORMODE of GLOBALONETOONE, if cell 6, face 2 in zone 9 should be connected to cell 1, face 4 in zone 10, the connections for zone 9 must include the line:

6 2 10 1 (cell# face#, connecting zone#, connecting cell#)

And the connections for zone 10 must include this line:

1 4 9 6

Global face neighbors are useful for telling Tecplot about the connections between zones. This could be used, for example, to smooth out the crease in Gouraud surface shading at zone boundaries. For cell-centered data, they can make contours and streamtraces more continuous at zone boundaries.

Cell numbering. For ordered (IJ or IJK) zones, cell numbers are defined by the index value of the first node, where Index = I + (J-1)*|MAXI| + (K-1)*|MAXI|*|MAXJ|. Because the number of nodes in each direction is one greater than the number of cells in that direction, there is no cell to correspond with the last point in each row. In the example below, there is no cell numbered "3", yet the first cell in the second row is numbered "4". As you define face neighbors, it may help you to think of a "ghost cell" at the end of each row (where I = MaxI) and at the end of each column in 3D (where J = MaxJ). If you probe any cell, the Face Neighbor tab of the Probe dialog will show the correct cell number.
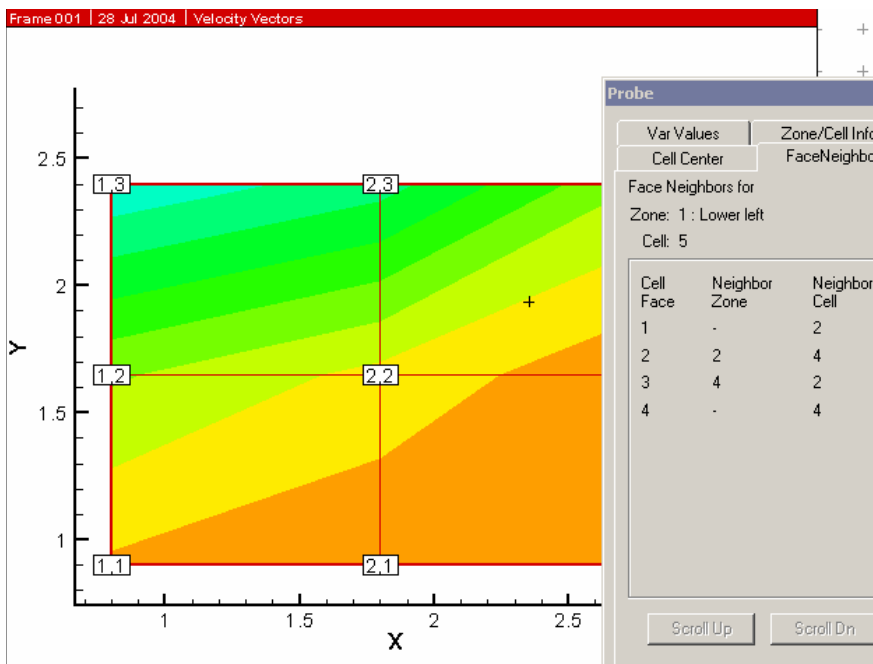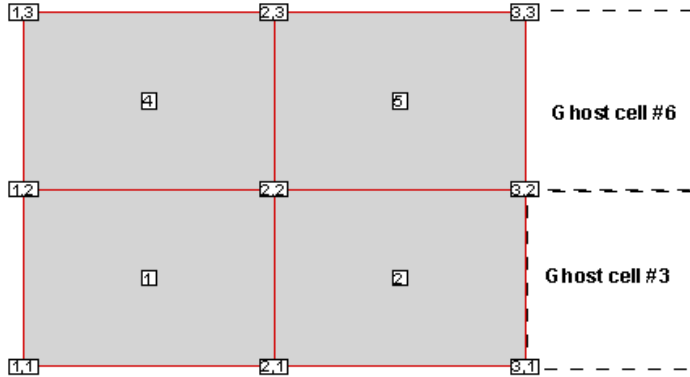
Zone Types and Control Lines. As stated above, there are two distinct types of zones: ordered zones and finite-element zones. Ordered zones are I-, IJ-, and IJK-ordered zones (**ZONE-**

TYPE=ORDERED). Finite-element zones are FE-line, -surface and -volume zones (**ZONETYPE** of **FELINESEG**, **FETRIANGLE**, **FEQUADRILATERAL**, **FETETRAHEDRON**, and **FEBRICK**). The control lines for these zone types differ in the parameters needed. Both zone types can use the **C** (*color*), **T** (*zonetitle*), **DATAPACKING**, **VARLOCATION**, **AUXDATA**, and **DT** (*datatype*) parameters.

The **T** parameter specifies a title for the zone. This may be any text string up to 64 characters in length. If you supply a longer text string, it is automatically truncated to the first 64 characters. The titles of zones appear in the **Zone Style** and other dialogs, and, optionally, in the XY- plot legend. (You can use keywords in the zone titles to identify sets of zones to enable/disable or to change zone attributes.) The **C** parameter sets an initial color for the zone. This may be overridden interactively, or by use of a stylesheet. The **DT** (*type1, type2, type3, ...*) parameter specifies the data types for the variables in a zone.

For ordered zones, you may specify the **I** (*IMax*), **J** (*JMax*), and **K** (*KMax*) parameters, which store the number of data points in the I, J, and K directions. **J** and **K** both default to 1. **I** must be specified if **J** is used; **I** and **J** must be specified if **K** is used. If all are omitted, Tecplot assumes an I-ordered zone and calculates *IMax* for you.

Note: I and **J** are not equivalent to either the number of variables or the number of data points. The number of data points is equal to the product of **I**, **J**, and **K**.

For finite-element zones, described in Section 2 - 4, "Finite-Element Data," you must specify the **N** (*numnodes*) and may optionally include **E** (*numelements*), and or the **NV** (*nodevalue*) parameter. If the **E** parameter is not specified, Tecplot calculates it from the number of node sets in the connectivity list following the node data. The **NV** (*nodevalue*) parameter specifies the number of the variables representing the "Node" value in finite-element data. The **NV** parameter is used infrequently, mostly when the order in which nodes are listed in the data file do not match the node numbering desired in the plot.

Section 2 - 3, "Ordered Data," provides examples of zone data in various formats, as well as sample pieces of FORTRAN code that you can use as templates to print out your own data. Our sample code is intended only as a general example—the zone data that it produces contains only one value per line. You may want to modify the code to suit your own needs.

The SOLUTIONTIME Parameter. Each zone can optionally specify a floating point time value representing it's solution time. Zones can be organized together by associating themselves to the same **STRANDID**.

The STRANDID Parameter. Each zone can optionally specify an integer value associating itself with a particular strand. More than one zone can associate itself with a particular strand and differentiate itself from other zones by assigning different **SOLUTIONTIME** values. StrandID's must

be positive integer values greater than or equal to 1. By convention strandID's are successive integer values.

Text Record

Text records are used to import text directly from a data file. Text can also be imported into Tecplot using a macro file. Text may be titles, labels, or other information. You may create data files containing only text records and read them into Tecplot just as you would read any other data file. You may delete and edit text originating from data files just like text created interactively.

The text record consists of a single control line. The control line starts with the keyword **TEXT** and has one or more options:

- The text string is defined in the required **T** (*text*) parameter.

- The color is controlled by the **C** (*color*) parameter.

- Use the **CS** (*coordinatesys*) parameter to specify the text coordinate system, either **FRAME**, **GRID** or **GRID3D.** If you specify the frame coordinate system (the default), the values of the **X** (*xorigin*) and **Y** (*yorigin*) parameters are in frame units; if you specify grid coordinates, **X** and **Y** are in grid units (that is, units of the physical coordinate system). **X** and **Y** locate the anchor point of the text string. For Polar Line plots, you may specify **THETA** and **R** instead of **X** and **Y**. Specify **X**, **Y** and **Z** for **GRID3D** coordinates.

- Use the **AN** (*textanchor*) parameter to specify the position of the anchor point relative to the text. There are nine possible anchor positions, as shown in .
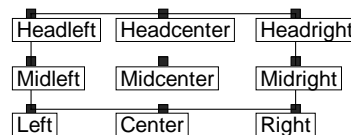
Figure 2-2. Text anchor positions—values for the **AN** parameter.

- Use the **HU** (*heightunits*) parameter to assign units for character heights. If the **CS** parameter is **FRAME**, you can set **HU** to either **FRAME** or **POINT**. If the **CS** parameter is **GRID**, you can set **HU** to either **GRID** or **FRAME**.

- Use the **H** parameter to specify the height; it is measured in the units defined by the **HU** parameter.

- To include multiple lines of text in a single text record, include **\\n** in the text string to indicate a new line.

- You can assign the line spacing for multi-line text using the **LS** (*linespacing*) parameter. The default value, 1, gives single-spacing. Use 1.5 for line-and-a-half spacing, 2 for double-spacing, and so on.

Optionally, you may draw a box around the text string using the **BX** (*boxtype*) parameter. The parameters **BXO** (*boxoutlinecolor*), **BXM** (*boxmargin*), and **LT** (*linethickness*) are used if the *boxtype* is **HOLLOW** or **FILLED**. The parameter **BXF** (*boxfillcolor*) is used only if the *boxtype* is **FILLED**. The default *boxtype*, **NOBOX**, ignores all other *box* parameters.

The **S** (*scope*) parameter specifies the text scope. **GLOBAL** scope is the same as selecting the check box Show in "Like" Frames in the Text Options dialog.

You may also use the **ZN** (*zone*) parameter to attach text to a specific zone or XY mapping. For further information, see Section 16.1.6.4, "Attaching Text to Zones or X-Y Mapping."

Text Record Examples. You may attach a macro command to the text with the **MFC** parameter. See Section 21 - 5 "Text and Geometry Links to Macros" on page 434 in the Tecplot 360 User's Manual.

Some simple examples of text records are shown below. The first text record specifies only the origin and the text. The next text record specifies the origin, color, font, and the text. The third text record specifies the origin, height, box attributes, and text. Note that the control line for the text can span multiple file lines if necessary (as in the third text record below). The last text record is an example of using 3D text in Tecplot.

```
TEXT X=50, Y=50, T="Example Text"

TEXT X=10, Y=10, F=TIMES-BOLD, C=BLUE, T="Blue Text"

TEXT X=25, Y=90, CS=FRAME, HU=POINT, H=14,
    BX=FILLED, BXF=YELLOW, BXO=BLACK, LS=1.5,
    T="Box Text \\n Multi-lined text"
```

```
    TEXT CS=GRID3D, X=0.23,Y=0.23,Z=0.5, T="Well 1"
```

Geometry Record

Geometry records are used to import geometries from a data file. Geometries are line drawings that may be boundaries, arrows, or even representations of physical structures. You may create data files containing only geometry and text records and read them into Tecplot. You may delete and edit geometries originating from data files just like the geometries that you create interactively.

The geometry record control line begins with the keyword **GEOMETRY**. Use the **CS** (*coordinatesys*) parameter to specify the geometry coordinate system, either **FRAME** or **GRID**. If you specify the frame coordinate system (the default), the values of the **X** (*xorigin*) and **Y** (*yorigin*) parameters are in frame units; if you specify grid coordinates, **X** and **Y** are in grid units (that is, units of the physical coordinate system). For Polar Line plots, you may specify **THETA** and **R** for **X** and **Y**. **X** and **Y** (or **THETA** and **R**) locate the anchor point, or origin, of the geometry, which is the center of a circle or ellipse, the lower left corner of a square or rectangle, and the anchor point of a polyline. The anchor point specifies the offset of all the points: if *X=1*, *Y=1*, and the first point is (1, 2), and the second point is (2, 4), then Tecplot draws at (2, 3) (1+1, 2+1) then (3, 5) (2+1, 4+1). In other words, the points for any geometry are always relative to the specified anchor point. The **Z** (*zorigin*) is specified only for **LINE3D** geometries, and, since **LINE3D** geometries are always in grid mode, **Z** is always in units of the Z-axis.

Geometry types are selected with the **T** (*geomtype*) parameter. The available geometry types are listed below:

- **SQUARE** - A square with lower left corner at **X**, **Y**.

- **RECTANGLE** - A rectangle with lower left corner at **X**, **Y**.

- **CIRCLE** - A circle centered at **X**, **Y**.

- **ELLIPSE** - An ellipse centered at **X**, **Y**.

- **LINE** - A set of 2D polylines (referred to as multi-polylines) anchored at **X**, **Y**.

- **LINE3D** - A set of 3D polylines (referred to as multi-polylines) anchored at
  X, **Y**, **Z**.

The color of the geometry is controlled by the **C** (*color*) parameter. Any geometry type except **LINE3D** may be filled with a color by using the **FC** (*fillcolor*) parameter. With both **C** (*color*) and **FC** (*fillcolor*) on the control line, the geometry is outlined in one color and filled with another. Each polyline of a **LINE** geometry is filled individually (by connecting the last point of the polyline with

the first). Not specifying the **FC** (*fillcolor*) parameter results in a hollow, or outlined, geometry drawn in the color of the **C** (*color*) parameter.

You can control how geometries are drawn using the **L** (*linetype*), **LT** (*linethickness*), and **PL** (*patternlength*) parameters. You can set **L** to any of Tecplot's line patterns (**SOLID**, **DASHED**, **DOTTED**, **DASHDOT**, **LONGDASH**, **DASHDOTDOT**). You can set **LT** and **PL** to any value, using frame units.

The control line of the geometry is followed by geometry data. For **SQUARE**, the geometry data consists of just one number: the side length of the square.

For **RECTANGLE**, the geometry data consists of two numbers: the first is the width (horizontal axis dimension), and the second is the height (vertical axis dimension).

For **CIRCLE**, the geometry data is one number: the radius. For **ELLIPSE**, the geometry data consists of two numbers: the first is the horizontal axis length and the second is the vertical axis length. For both circles and ellipses, you can use the **EP** (*numellipsepts*) parameter to specify the number of points used to draw circles and ellipses. All computer-generated curves are simply collections of very short line segments; the **EP** parameter allows you to control how many line segments Tecplot uses to approximate circles and ellipses. The default is 72.

For **LINE** and **LINE3D** geometries, the geometry data is controlled by the **F** (*format*) parameter. These geometries may be specified in either **POINT** or **BLOCK** format. By default, **POINT** format is assumed. Each geometry is specified by the total number of polylines, up to a maximum of 50. Each polyline is defined by a number of points and a series of XY- or XYZ- coordinate points between which the line segments are drawn. In **POINT** format, the XY- or XYZ-coordinates are given together for each point. In **BLOCK** format, all the X-values are listed, then all the Y-values, and (for **LINE3D** geometries) all the Z-values. All coordinates are relative to the **X**, **Y**, and **Z** specified on the control line. You can specify points in either single or double precision by setting the **DT** (*datatype*) parameter to either **SINGLE** or **DOUBLE**.

For **LINE** geometries, you can specify arrowheads using the **AAT** (*arrowheadattach*), **AST** (*arrowheadstyle*), **ASZ** (*arrowheadsize*), and **AAN** (*arrowheadangle*) parameters. See Section , "Summary of Data File Records,"for details. These parameters provide the same functionality available when you create a line geometry interactively.

The **S** (*scope*) parameter specifies the geometry's scope. **GLOBAL** scope is the same as selecting the check box Show in Like Frames in the Geometry dialog. See Geometry Details in the Tecplot 360 User's Manual for details.

You may also use the **ZN** (*zone*) parameter to attach geometry to a specific zone or XY-mapping.

You may attach a macro command to the text with the **MFC** parameter. See <u>Text & Geometry Links to Macos in the Tecplot User's Manual</u>.

LINE3D geometries must be created in a data file. They may not be created interactively. **LINE3D** geometries are always in grid mode. To view **LINE3D** geometries in Tecplot, your plot type must be in 3D Cartesian, which requires at least one zone. Thus, a data file with only **LINE3D** geometries is useful only as a supplement to other data files.

Geometry Record Examples. The following geometry record defines a rectangle of **40** width and **30** height:

```
GEOMETRY T=RECTANGLE
40 30
```

The following geometry record defines an origin and a red circle of **20** radius, with an origin of (**75**, **75**) that is filled with blue:

```
GEOMETRY X=75, Y=75, T=CIRCLE, C=RED, FC=BLUE,CS=FRAME
 20
```

The following geometry record defines an origin and two polylines, drawn using the Custom 3 color. The first polyline is composed of three points, the second of two points.

```
GEOMETRY X=50, Y=50, T=LINE, C=CUST3
 2
 3
 0 1
 0 0
 2 0
 2
 0 0
 1 2
```

In **BLOCK** format, the same geometry appears as:

```
GEOMETRY X=50, Y=50, T=LINE, C=CUST3, F=BLOCK, CS=FRAME
 2
 3
 0 0 2
 1 0 0
 2
 0 1
 0 2
```

The next geometry record defines a purple ellipse with a horizontal axis length of **20** and a vertical axis length of **10**, with an origin of (**10, 70**), that is filled with yellow.

```
GEOMETRY X=10, Y=70, T=ELLIPSE, C=PURPLE, FC=YELLOW
 20 10
```

The final geometry record is a 3D polyline with four points that is composed of one polyline using the default origin of (**0, 0, 0**):

```
GEOMETRY T=LINE3D
 1
 4
 0 0 0
 1 2 2
 3 2 3
 4 1 2
```

In **BLOCK** format, this geometry record can be written as follows:

```
GEOMETRY T=LINE3D, F=BLOCK
 1
 4
 0 1 3 4
 0 2 2 1
 0 2 3 2
```

A More Detailed Example of a Geometry Record

In the **TextGeom** file shown below, there are four text records (showing a circle, ellipse, rectangle, and line). A plot of the file is shown in Figure 2-3.
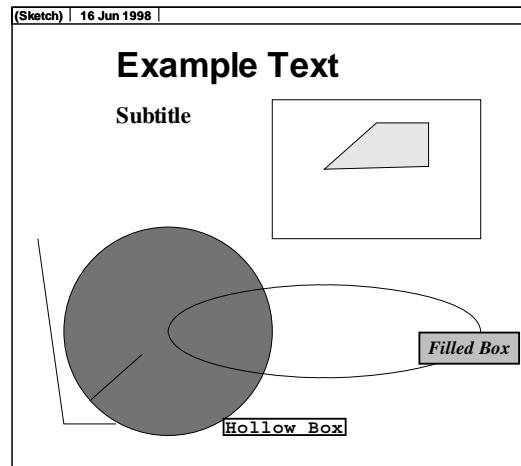
Figure 2-3. Text and geometries created from the sample in Section , "A More Detailed Example of a Geometry Record".

```
TEXT X=20, Y=85, F=HELV-BOLD, C=BLUE, H=7.5,
      T="Example Text"
TEXT X=20, Y=75, F=TIMES-BOLD, H=5, T="Subtitle"
TEXT X=80, Y=25, F=TIMES-ITALIC-BOLD, H=4, C=RED,
      BX=FILLED, BXF=YELLOW, BXM=50, BXO=CYAN,
      T="Filled Box"
TEXT X=41, Y=8, H=4, F=COURIER-BOLD,
      C=CUST3, BX=HOLLOW, BXO=CUST4, T="Hollow Box"
GEOMETRY X=50, Y=50, T=RECTANGLE, FC=WHITE, C=BLUE
   40 30
GEOMETRY X=30, Y=30, T=CIRCLE, FC=BLUE, C=GREEN
   20
GEOMETRY X=70, Y=65, T=LINE, FC=PURPLE, C=BLACK
   1
   4
   -10 0
```

```
    0 10
    010 10
    10 0.6
GEOMETRY T=LINE, C=CUST1
    2
    3
    5 50
    10 10
    20 10
    2
    15 15
    25 25
GEOMETRY X=60, Y=30, T=ELLIPSE, C=CUST8
    30 10
```

Custom Label Record

The custom label record is an optional record to define sets of text strings for use in custom labeling the values of an axis, contour legend or value labels, or variable-value node labels. The custom label record begins with the keyword **CUSTOMLABELS**, followed by one or more text strings. The text strings *must* be enclosed within double quotes ("**"**") if they contain any commas, spaces or other special characters, or if they might be confused with valid data file keywords. Enclosing the strings in double quotes is always recommended.

The first custom label string corresponds to a value of one on the axis, the next to a value of two, the next to a value of three, and so forth. Custom labels may appear one to a line, or there may be more than one on a line, separated by a comma or space. Multiple custom label records can be present in a data file. If this is the case, you choose which set to assign to a given axis, contour legend, or variable-value node labels. Custom labels are discussed in more detail in Using Custom Labels in the Tecplot 360 User's Manual.

A simple example of a custom-label record is shown below. **MON** corresponds to a value of **1**, **TUE** corresponds to **2**, **WED** to **3**, **THU** to **4**, and **FRI** to **5**. Since custom labels have a wrap-around effect, **MON** also corresponds to the values **6**, **11**, and so forth.

```
CUSTOMLABELS "MON", "TUE", "WED", "THU", "FRI"
```

Data Set Auxiliary Data Record

There is frequently auxiliary data (or Metadata) that helps describe the data set. For example, experimental data my have information about the facility and time at which the data was taken, and other parameter that describe the experiment. Likewise, simulation results have auxiliary data (such as reference quantities for non-dimensional data) needed to fully analyze and present the results. This data may be concerning the data set as a whole or it can vary from zone to zone. The ASCII file format for specifying auxiliary data associated with the data set are described here. The format for zonal auxiliary data is described in Section 4.1.2.7.

The data set auxiliary data control line is as follows:

```
DATASETAUXDATA name-string = "value string"
```

where name-string is a unique character string with no spaces. There may be multiple **DATASE-TAUXDATA** records but name-string must be unique for each one.

Auxiliary data may be used in text, macros, equations (if it is numeric), and accessed from add-ons. It may also be viewed directly in the AuxData page of the Data Set Information dialog.

Data Set Auxiliary Data Examples. The following auxiliary data contain flow field information that might be found in output from a computational fluid-dynamics simulation.

```
DATASETAUXDATA MachNo = "1.2"
DATASETAUXDATA Alpha = "5"
DATASETAUXDATA RefTemperature = "250"
DATASETAUXDATA RefPressure = "101325"
DATASETAUXDATA Configuration = "A2 No. 3"
DATASETAUXDATA Date = "August 5, 2003"
DATASETAUXDATA Region = "NE Quadrant of Sector 47"
```

You may then use the numerical values in equations to modify the variables like this:

```
{P} = {P_non_dim} * AuxDataSet:RefPressure
```

Configuration and **Date** may then be included in text on the plot. This makes it easier to automate your plotting tasks using layout files and/or macros.

Summary of Data File Records

The following table summarizes the records and parameters allowable in Tecplot data files.

`TITLE=`"*datasettitle*" `VARIABLES=`"**vname1**","**vname2**", ...	`FILE` `HEADER`

`ZONE`   `T=`"*zonetitle*", `I=`*imax*, `J=`*jmax*, `K=`*kmax*, `C=`*color*, `ZONETYPE=ORDERED`, `DT=`(*datatypelist*), `DATA-PACKING=`*datapacking*, `SOLUTIONTIME=`*time*, `STRANDID=`*strandid*,`PARENTZONE=`*parentzone*
 	
  `VARLOCATION=`([*varset*]=*varlocation*, [*varset*]=*var-location*), `AUXDATA` *auxvar*="**value**",
 	
`VARSHARELIST=`([*varset*]=*zzz*,[*varset*]=*zzz*)
 	
`FACENEIGHBORMODE=`*faceneighbormode*
 	
`FACENEIGHBORCONNECTLIST=`*faceneighborcon-nections*
 	
     --- DATA FOR ORDERED ZONE ---
	`ORDERED`
	`ZONE`
	`RECORD`

ZONE	T="*zonetitle*", N=*numnodes*, E=*numelements*, C=*color*, ZONETYPE=*feformat*, DT=(*datatypelist*), DATAPACKING=*datapacking*, NV=*nodevariable*, CONNECTIVITYSHAREZONE=*zzz*, STRAN-DID=*strandid*, SOLUTIONTIME=*time*, AUXDATA *auxvar*="**value**", PARENTZONE=*parentzone*	FINITE-ELEMENT ZONE RECORD
	VARLOCATION=([*varset*]=*varlocation*, [*varset*]=*var-location*), VARSHARELIST=([*varset*]=*zzz*,[*var-set*]=*zzz*)	
	FACENEIGHBORMODE=*faceneighbormode*	
	FACENEIGHBORCONNECTLIST=*faceneighborcon-nections*	
	--- DATA FOR FINITE-ELEMENT ZONE ---	

TEXT	X=*xorigin*,Y=*yorigin*, Z=*zorigin*, THETA=*thetaori-gin*,R=*rorigin*, F=*font*, CS=*coordinatesys*, HU=*heigh-tunits*, AN=*textanchor*, C=*color*, A=*angle*, H=*height*, S=*scope*, LS=*linespacing*, T="*text*", BX=*boxtype*, BXM=*boxmargin*, BXF=*boxfillcolor*, BXO=*boxcolor*, LT=*linethickness*, ZN=*zone*, CLIPPING=*clipping*, MFC=*macrofunction*	TEXT RECORD

GEOME-TRY	X=*xorigin*,Y=*yorigin*, Z=*zorigin*, THETA=*theta-origin*, R=*rorigin*, T=*geomtype*, CS=*coordinate-sys*, C=*color*, L=*linetype*, DT=*datatype*, PL=*patternlength*, LT=*linethickness*, EP=*numellipsepts*, AST=*arrowheadstyle*, AAT=*arrowheadattach*, ASZ=*arrowheadsize*, AAN=*arrowheadangle*, S=*scope*, F=*geomfor-mat*, FC=*geomfillcolor*, ZN=*zone*, MFC=*macro-function*, CLIPPING=*clipping*, DRAWORDER=*draworder*	GEOME-TRY RECORD
	---DATA FOR GEOMETRY RECORD---	

`DATASETAUX-` `DATA`	*auxvar*1="value1", *auxvar*2="value2", ...	DATA AUXILIARY RECORD
`CUSTOMLA-` `BELS`	*"label1", "label2", ...*	CUSTOM LABEL RECORD
`VARAUXDATA`	v1 *auxvar*="value", v2 *auxvar*="value", ...	VARIABLE AUXILIARY RECORD

2 - 2 ASCII Data File Parameters

angle	**angle in degrees counter-clockwise from horizontal**
arrowheadstyle	`PLAIN, HOLLOW, FILLED`
arrowheadattach	`NONE, BEGINNING, END, BOTH`
arrowheadsize	**size of arrowhead in frame units**
arrowheadangle	**angle of arrowhead in degrees**
auxvar	**name of auxiliary data variable**
boxcolor	**fill color for text box use color options**
boxfillcolor	**fill color for text in box as fraction of text height**
boxmargin	**margin around text in box as fraction of text height**
boxtype	`NOBOX, HOLLOW ,FILLED`
clipping	`CLIPTOVIEWPORT, CLIPTOFRAME`
color	`BLACK, RED, GREEN, BLUE, CYAN, YELLOW, PURPLE,` `WHITE, CUST1, ..., CUST8`
coordinatesys	`FRAME, GRID, GRID3D`
datapacking	`BLOCK, POINT`
datasettitle	**title of dataset**
datatype	`SINGLE, DOUBLE`
datatypelist	`SINGLE, DOUBLE, LONGINT, SHORTINT, BYTE, BIT`
draworder	`AFTERDATA,BEFOREDATA`
faceneighborconnections	*Faceneighborconnectlist* **starts with the cell and cell-face affected by the connection. It then contains neighboring cells and zones depending on the** `FACENEIGHBORMODE.`
faceneighbormode	`LOCALONETOONE, LOCALONETOMANY, GLOBALONE-` `TOONE, GLOBALONETOMANY`
feformat	`FELINESEG, FETRIANGLE, FEQUADRILATERAL,` `FETETRAHEDRON, FEBRICK`

font	`HELV, HELV-BOLD, TIMES, TIMES-ITALIC, TIMES-`
	`BOLD, TIMES-ITALIC-BOLD, COURIER, COURIER-`
	`BOLD, GREEK, MATH, USER-DEF`

geomfillcolor	**fill color for geomtry use color options**

geomformat	`POINT`, `BLOCK`
geomtype	`LINE, SQUARE, RECTANGLE, CIRCLE, ELLIPSE`
height	text height in frame units
heightunits	In `FRAME` coordinatesys either `FRAME` or `POINT`; in `GRID` coordinatesys either `GRID` or `FRAME`
imax, jmax, kmax	number of points in the I- J- or K-direction
labelN	string for value of N when using custom labels
linespacing	line spacing for multiple-line text
linethickness	Thickness of text box or geometry outline
linetype	`SOLID, DASHED, DASHDOT, DOTTED, LONGDASH, DASHDOTDOT`
macrofunction	macro function command
nodevariable	number of the variable representing the "Node" value
numelements	number of elements in finite-element zone
numellipsepts	number of points used to approximate circles or ellipses
numnodes	number of nodes in finite-element zone
orderedformat	`BLOCK, POINT`
parentzone	Ones-based parent zone number within the dataset. A zone may not specify itself as its parent.
patternlength	pattern length for linetype
rorigin	r origin of the object in *coordinatesys* units, Polar plots only
scope	`GLOBAL, LOCAL`
strandid	integer value associating the zone with a given strand
text	alphanumeric text string
textanchor	`LEFT, CENTER, RIGHT, MIDLEFT, MIDCENTER, MIDRIGHT, HEAD-LEFT, HEADCENTER, HEADRIGHT`
thetaorigin	theta origin of the object in *coordinatesys* units, Polar plots only
time	floating point time value for the zone
varlocation	`NODAL, CELLCENTERED`
varset	list of variables to use

vN	**number of the Nth variable**
xorigin, yorigin	**x or y origin of the object in coordinatesys units**
zorigin	**z origin of object (always in `GRID` units)**
zone	**zone number to which this item is assigned (0=all)**
zonetitle	**title of zone**
zzz	**The source zone for shared variables. If omitted, the variables are shared from the previous zone**

2 - 3 Ordered Data

For ordered data, the numerical values in the zone data must be in either **POINT** or **BLOCK** format, specified by the **DATAPACKING** parameter.

2- 3.1 I-Ordered Data

I-ordered data has only one index, the I-index. This type of data is typically used for XY-plots, scatter plots, and irregular (random) data for triangulation or for interpolation into an IJ-or IJK-ordered zone within Tecplot.

In I-ordered data, the I-index varies from one to *IMax*. The total number of data points is *IMax*. For zones with only nodal variables, the total number of values in the zone data is *IMax*N* (where *N* is the number of variables). For a mixture of nodal and cell-centered variables, the number of values in the zone data is *IMax*Nn+(IMax-1)*Nc*, where *Nn* is the number of nodal variables and *Nc* is the number of cell-centered variables. For data in **POINT** format, *IMax* is calculated by Tecplot from the zone data if it is not explicitly set by the zone control line (using the **I-** parameter).

I-Ordered Data in POINT Format Example. A simple example of I-ordered data in **POINT** format is listed below. There are two variables (**X**, **Y**) and five data points. In this example,

each row of data corresponds to a data point and each column to a variable. This data set is plotted in Figure 2-4; each data point is labeled with its I-index.
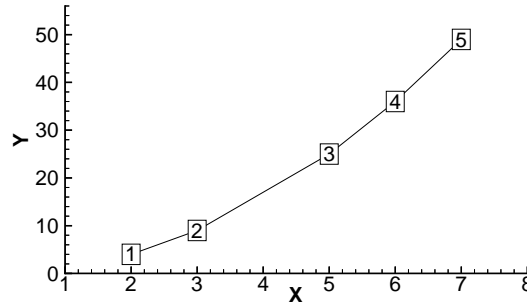

Figure 2-4. An I-ordered data set.

```
VARIABLES = "X","Y"
ZONE I=5, DATAPACKING=POINT
2    4
3    9
5    25
6    36
7    49
```

For this data try omitting the **VARIABLES** and **ZONE** lines, leaving two columns of information. In this case, Tecplot would count the columns to determine the number of variables, count rows to determine I-dimension, and label the variables V1 and V2.

FORTRAN Code Example to Generate I-Ordered Data in POINT Format.

The following sample FORTRAN code shows how to create I-ordered data in **POINT** format:

```
INTEGER VAR
    .
    .
    .
WRITE (*,*) ´ZONE DATAPACKING=POINT, I=´, IMAX
DO 1 I=1,IMAX
    DO 1 VAR=1,NUMVAR
1       WRITE (*,*) ARRAY(VAR,I)
```

I-Ordered Data in BLOCK Format Example. The data from Section <u>"I-Ordered Data in POINT Format Example" on page 32</u> is shown below in **BLOCK** format. In this example, each column of zone data corresponds to a data point; each row to a variable.

```
VARIABLES = "X", "Y"
ZONE I=5, DATAPACKING=BLOCK
2 3 5 6 7
4 9 25 36 49
```

In **BLOCK** format all *IMax* values of each variable are listed, one variable at a time.

FORTRAN Code to Generate I-Ordered Data in BLOCK Format Example.

The following sample FORTRAN code shows how to create I-ordered data in **BLOCK** format:

```
INTEGER VAR
   .
   .
   .
WRITE (*,*) ´ZONE DATAPACKING=BLOCK, I=´, IMAX
DO 1 VAR=1,NUMVAR
    DO 1 I=1,IMAX
1       WRITE (*,*) ARRAY(VAR,I)
```

Multi-Zone XY Line Plot Example. The two tables below show the values of pressure and temperature measured at four locations on some object at two different times. The four locations are different for each time measurement.

Time = 0.0 seconds:

Position	Temperature	Pressure
71.30	563.7	101362.5
86.70	556.7	101349.6
103.1	540.8	101345.4
124.4	449.2	101345.2

Figure 2-5.

Time = 0.1 seconds:

Position	Temperature	Pressure
71.31	564.9	101362.1
84.42	553.1	101348.9
103.1	540.5	101344.0
124.8	458.5	101342.2

For this case, we want to set up two zones in the data file, one for each time value. Each zone has three variables (**Position**, **Temperature**, and **Pressure**) and four data points (one for each location). This means that *IMax=4* for each zone. We include a text record (discussed in Section

"Text Record") to add a title to the plot. A data file in **POINT** format is given below. The plot shown in Figure 2-6 can be produced from this file.
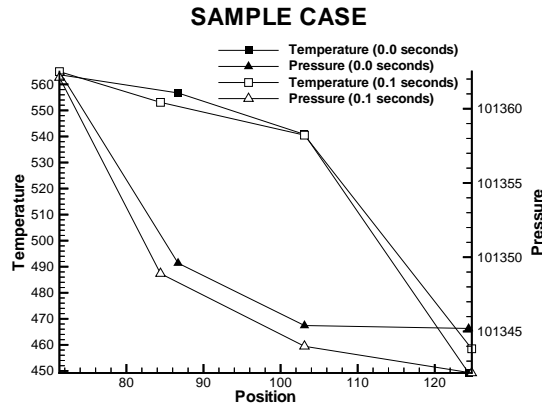

Figure 2-6. A multi-zone XY Line plot.

```
TITLE = "Example: Multi-Zone XY Line Plot"
VARIABLES = "Position", "Temperature", "Pressure"
ZONE T="0.0 seconds", I=4
71.30 563.7 101362.5
86.70 556.7 101349.6
103.1 540.8 101345.4
124.4 449.2 101345.2
ZONE T="0.1 seconds", I=4
71.31 564.9 101362.1
84.42 553.1 101348.9
103.1 540.5 101344.0
124.8 458.5 101342.2
TEXT CS=FRAME, HU=POINT, X=16, Y=90, H=28, T="SAMPLE CASE"
```

A data file in **BLOCK** format is shown below. All of the values for the first variable (**Position**) at each data point are listed first, then all of the values for the second variable (**Temperature**) at each data point, and so forth.

```
TITLE = "Example: Multi-Zone XY Line Plot"
VARIABLES = "Position", "Temperature", "Pressure"
```

```
ZONE DATAPACKING=BLOCK, T="0.0 seconds", I=4
71.30 86.70 103.1 124.4
563.7 556.7 540.8 449.2
101362.5 101349.6 101345.4 101345.2
ZONE DATAPACKING=BLOCK, T="0.1 seconds", I=4
71.31 84.42 103.1 124.8
564.9 553.1 540.5 458.5
101362.1 101348.9 101344.0 101342.2
TEXT CS=FRAME, HU=POINT, X=16, Y=90, H=28, T="SAMPLE CASE"
```

A more compact data file for this example is in the point format shown below. Tecplot determines the number of variables from the number of values in the first line of data under the first zone. The variables and zones are assigned default names.

```
ZONE
71.30 563.7 101362.5
86.70 556.7 101349.6 103.1 540.8 101345.4 124.4 449.2 101345.2
ZONE
71.31 564.9 101362.1 84.42 553.1 101348.9 103.1 540.5 101344.0
124.8 458.5 101342.2
TEXT CS=FRAME, HU=POINT, X=16, Y=90, H=28, T="SAMPLE CASE"
```

Multi-Zone XY Line Plot with Variable Sharing Example. If the data from the section above was taken at the same position for both times, variable sharing could reduce memory usage and file size. That file appears as:

```
TITLE = "Example: Multi-Zone XY Line Plot with Variable Sharing"
VARIABLES = "Position", "Temperature", "Pressure"
ZONE T="0.0 seconds", I=4
71.30 563.7 101362.5
86.70 556.7 101349.6
103.1 540.8 101345.4
124.4 449.2 101345.2
ZONE T="0.1 seconds", I=4, VARSHARELIST=([1]=1)
564.9 101362.1
553.1 101348.9
540.5 101344.0
458.5 101342.2
TEXT CS=FRAME, HU=POINT, X=16, Y=90, H=28, T="SAMPLE VARIABLE
SHARING CASE"
```

2- 3.2 IJ-Ordered Data

IJ-ordered data has two indices: I and J. IJ-ordered data is typically used for 2D and 3D surface mesh, contour, vector, and shade plots, but it can also be used to plot families of lines in XY-plots. Refer to the *Tecplot 360 User's Manual* for more information on data structure. In IJ-ordered data, the I-index varies from 1 to *IMax*, and the J-index varies from one to *JMax*. The total number of data points (nodes) is *IMax*JMax*. For zones with only nodal variables, the total number of numerical values in the zone data is *IMax*JMax*N* (where *N* is the number of variables). For a mixture of nodal and cell-centered variables, the number of values in the zone data is *IMax*JMax*Nn+(IMax-1)*(JMax-1)*Nc*, where *Nn* is the number of nodal variables and *Nc* is the number of cell-centered variables. Both *IMax* and *JMax* must be specified in the zone control line (with the **I** and **J** parameters). The I- and J-indices should not be confused with the X- and Y-coordinates—on occasions the two may coincide, but this is not the typical case.

The I-index varies the fastest. That is, when you write programs to print IJ-ordered data, the I-index is the inner loop and the J-index is the outer loop. Note the similarity between I-ordered data and IJ-ordered data with *JMax=1*.

IJ-Ordered Data in POINT Format Example. An example of IJ-ordered data in **POINT** format is listed below. There are four variables (**X**, **Y**, **Temperature**, **Pressure**) and six
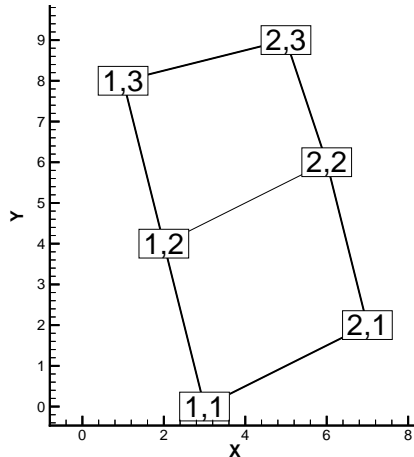

Figure 2-7. An IJ-ordered data set.

data points. In this example, each row of data corresponds to a data point; each column to a variable. The first two lines are for *J=1*, the next two for *J=2*, the last two for *J=3*. The first, third, and fifth lines are for *I=1*; the second, fourth, and sixth lines are for *I=2*. This data is plotted in Figure 2-7; each data point is labeled with its IJ-index.

```
VARIABLES = "X", "Y", "Temperature", "Pressure"
ZONE I=2, J=3, DATAPACKING=POINT
3 0 0 50
7 2 0 43
2 4 1 42
6 6 0 37
1 8 1 30
5 9 1 21
```

FORTRAN Code to Generate IJ-Ordered Data in POINT Format Example.

The following sample FORTRAN code shows how to create IJ-ordered data in **POINT** format:

```
WRITE (*,*) ´VARIABLES = "X", "Y", "Temperature", "Pressure"´
WRITE (*,*) ´ZONE I=´, IMAX,´, J=´, JMAX,´,´DATAPACKING=POINT´
 DO 1 J=1,JMAX
    DO 1 I=1, IMAX
1          WRITE (*,*) X(I,J), Y(I,J), T(I,J), P(I,J)
```

IJ-Ordered Data Set in BLOCK Format Example. The same data set as in Section <u>, "IJ-Ordered Data in POINT Format Example,"</u> is shown in **BLOCK** format below. In this example, each column of data corresponds to a data point; each row to a variable.

```
VARIABLES = "X", "Y", "Temperature", "Pressure"
ZONE I=2, J=3, DATAPACKING=BLOCK
3 7 2 6 1 5
0 2 4 6 8 9
0 0 1 0 1 1
50 43 42 37 30 21
```

In **BLOCK** format, all *IMax*JMax* values of each variable are listed, one variable at a time. Within each variable block, all the values of a variable at each data point are listed.

FORTRAN Code to Generate IJ-Ordered Data in BLOCK Format Example.

The following sample FORTRAN code shows how to create IJ-ordered data in **BLOCK** format:

```
INTEGER VAR
   .
   .
   .
WRITE (*,*) ´ZONE DATAPACKING=BLOCK, I=´, IMAX, ´, J=´, JMAX
DO 1 VAR=1,NUMVAR
```

```
        DO 1 J=1,JMAX
            DO 1 I=1,IMAX
1               WRITE (*,*) ARRAY(VAR,I,J)
```

IJ-Ordered Data with Cell-Centered Data. An example of IJ-ordered data with cell-centered variables might include four variables (**X**, **Y**, **Temperature**, **Pressure**), nine data points, and four cells where **Temperature** and **Pressure** are cell-centered.
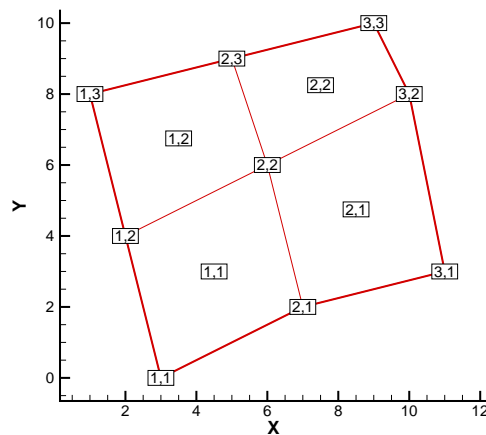

Figure 2-8. An IJ-ordered data set with cell-centered data.

```
VARIABLES = "X", "Y", "Temperature", "Pressure"
ZONE I=3, J=3, DATAPACKING=BLOCK, VARLOCATION=(3=CELLCENTERED,
4=CELLCENTERED)
3 7 11 2 6 10 1 5 9
0 2 3 4 6 8 8 9 10
0 2 1 3
45 60 35 70
```

The nodal variables of **X** and **Y** are specified at all nine nodes, and the values of cell-centered variables are specified at the four cells **[(IMax-1)*(JMax-1)]**. Zones with cell-centered data must have **DATAPACKING=BLOCK**.

Two-Dimensional Field Plots

A 2D field plot typically uses an IJ-ordered or finite-element surface data set. However, any data structure can be viewed as a 2D field plot, by simply selecting "2D Cartesian" from the plot-type menu in the Sidebar.

An IJ-ordered data file has the basic structure shown below:

```
TITLE = "Example: Multi-Zone 2D Plot"
VARIABLES = "X", "Y", "Press", "Temp", "Vel"
ZONE T="BIG ZONE", I=3, J=3, DATAPACKING=POINT
1.0 2.0 100.0 50.0 1.0
1.0 3.0 95.0 50.0 1.00
1.0 4.0 90.0 50.0 0.90
2.0 2.0 91.0 40.0 0.90
2.0 3.0 85.0 40.0 0.90
2.0 4.0 80.0 40.0 0.80
3.0 2.0 89.0 35.0 0.85
3.0 3.0 83.0 35.0 0.80
3.0 4.0 79.0 35.0 0.80
ZONE T="SMALL ZONE", I=3, J=2, DATAPACKING=POINT
3.0 2.0 89.0 35.0 0.85
3.5 2.0 80.0 35.0 0.85
4.0 2.0 78.0 35.0 0.80
3.0 3.0 83.0 35.0 0.80
3.5 3.0 80.0 35.0 0.85
4.0 3.0 77.0 33.0 0.78
```

This data file has two zones and five variables, and is included with Tecplot as the file **examples/dat/multzn2d.dat**. The first zone has nine data points arranged in a three-by-three grid (I=3, J=3). Each row of each zone represents one data point, where each column corresponds to the value of each variable for a given data point, i.e. X = 1.0, Y = 2.0, Press = 100.0, Temp = 50..0, and Vel - 1.0 for data point 1 in zone one (Big Zone).

Similarly, the second zone (Small Zone) has six data points in a three-by-two mesh (I=3, J=2). Reading this data file yields the mesh plot shown in Figure 2-9. A 2D finite-element data file is shown below (included in your Tecplot distribution as **examples/dat/2dfed.dat**):.
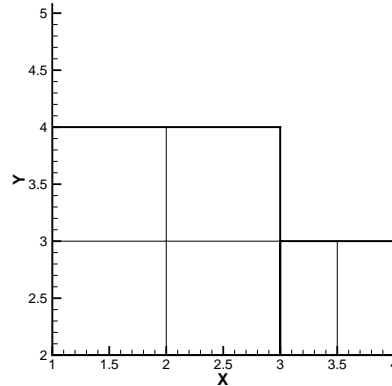

Figure 2-9. A 2D field plot.

```
TITLE = "Example: 2D Finite-Element Data"
VARIABLES = "X", "Y", "P", "T"
ZONE N=8, E=4, DATAPACKING=POINT, ZONETYPE=FEQUADRILATERAL
0.0 1.0 75.0 1.6
1.0 1.0 100.0 1.5
3.0 1.0 300.0 2.0
0.0 0.0 50.0 1.0
1.0 0.0 100.0 1.4
3.0 0.0 200.0 2.2
4.0 0.0 400.0 3.0
2.0 2.0 280.0 1.9
1 2 5 4
2 3 6 5
6 7 3 3
3 2 8 8
```

The above finite-element data file has eight nodes (the first 8 rows of the zone) and four elements (the last four rows of the zone). The each row in the node matrix represents a given node. Each column in the row matrix corresponds to the value of each variable in at a given node. The order of the variables definition correlates to the order the variables are named in the data set, i.e. for node 1, X = 0.0, Y=1.0, P = 75.0 and T = 1.6.

The element matrix defines the connectivity of the nodes, i.e. element one is composed of nodes 1,2,5 and 4. The data set yields the simple mesh plot shown in Figure 2-10.

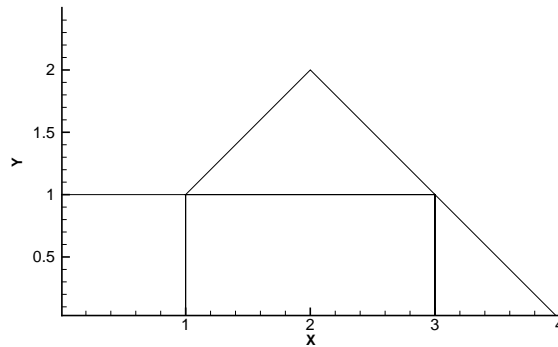**Figure 2-10.** A 2D mesh plot of a finite-element data set.

Please refer to Data Structure in the Tecplot 360 User's Manual for information on ordered and FE data sets and Chapter 2 "ASCII Data" for information on formatting your data sets as ordered or FE.

Three-Dimensional Field Plots

Creating a 3D field plot from a 2D plot is usually as simple as selecting 3D Cartesian from the plot type dropdown on the sidebar.

3D Cartesian is the default plot type for 3D volume (IJK-ordered and FE-volume) data sets

IJK-ordered data sets have the general form shown below:

```
TITLE = "Example: Simple 3D Volume Data"
 VARIABLES = "X", "Y", "Z", "Density"
 ZONE I=3, J=4, K=3, DATAPACKING=POINT
 1.0 2.0 1.1 2.21
 2.0 2.1 1.2 5.05
 3.0 2.2 1.1 7.16
 1.0 3.0 1.2 3.66
...
```

The complete ASCII data file is included with Tecplot as **`simp3dpt.dat`** (POINT format), and in block format as **`simp3dbk.dat`**. When you read either of these files into Tecplot, you immediately get the plot shown in .
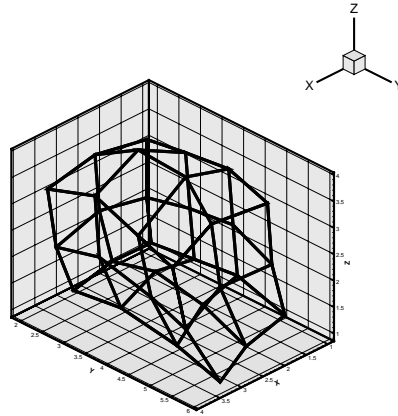

Figure 2-11. Plot of a 3D volume.

Finite-element volume data sets, like FE-surface data sets, consist of two separate lists—the value list and the connectivity list. A portion of a finite-element volume data file is shown below:

The full data file, consisting of a single FE-Brick zone, is included with Tecplot as **`febrfep.dat`** (POINT format), and in BLOCK format as **`febrfeb.dat`**.

```
TITLE = "Example: FE-Volume Brick Data"
 VARIABLES = "X", "Y", "Z", "Temperature"
 ZONE N=14, E=5, DATAPACKING=POINT, ZONETYPE=FEBRICK
 0.0 0.0 0.0 9.5
 1.0 1.0 0.0 14.5
 1.0 0.0 0.0 15.0
 1.0 1.0 1.0 16.0
...
 1 1 1 1 2 4 5 3
 2 4 5 3 7 10 11 8
 4 4 5 5 10 13 14 11
 4 4 4 4 9 12 13 10
 2 2 4 4 7 6 9 10
```

The above finite-element data file has fourteen nodes (the first four are displayed) and five elements (the last five rows of the zone). The each row in the node matrix represents a given node. Each column in the row matrix corresponds to the value of each variable in at a given node. The order of the variables definition correlates to the order the variables are names in the data set, i.e. for node 1, X = 0.0, Y=0.0, Z=0.0 and Temperature = 9.5.

The element matrix defines the connectivity of the nodes, i.e. element one is composed of nodes 1,1,1,1,2,4,5,3. Node 1 is repeated several times because the FEBRICK zone type requires eight nodes for each element. If 8 unique nodes are not present in physical space, one node is repeated to fulfill the requirement.

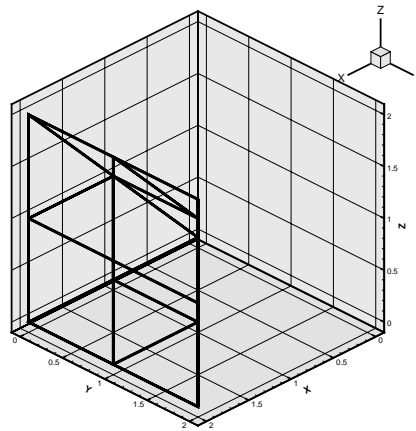When you read either of these files into Tecplot, you obtain the plot shown in Figure 2-12.

Figure 2-12. A 3D field plot of a finite-element volume data set.

2- 3.3 IJK-Ordered Data

IJK-ordered data has three indices: I, J, and K. This type of data is typically used for 3D volume plots, although planes of the data can be used for 2D and 3D surface plots. See Chapter Ordered Data in the Tecplot 360 User's Manual for more information.

In IJK-ordered data, the I-index varies from 1 to *IMax*, the J-index varies from one to *JMax*, and the K-index varies from one to *KMax*. The total number of data points (nodes) is *IMax*JMax*KMax*. For zones with only nodal variables the total number of values in the zone data is *IMax*JMax*KMax*N*, where *N* is the number of variables. For a mixture of nodal and cell-centered variables, the number of values in the zone data is *IMax*JMax*KMax*Nn+(IMax-1)*(JMax-*

1)*(KMax-1)*Nc, where *Nn* is the number of nodal variables and *Nc* is the number of cell-centered variables. The three indices, *IMax*, *JMax*, and *KMax*, must be specified in the zone control line using the **I**-, **J**-, and **K**-parameters.

The I-index varies the fastest; the J-index the next fastest; the K-index the slowest. If you write a program to print IJK-ordered data, the I-index is the inner loop, the K-index is the outer loop, and the J-index is the loop in between. Note the similarity between IJ-ordered data and IJK-ordered data with *KMax=1*.

IJK-Ordered Data in POINT Format Example. An example of IJK-ordered data in **POINT** format is listed below. There are four variables (**X**, **Y**, **Z**, **Temperature**) and twelve data points. Each row of data corresponds to a data point; each column to a variable. This data is plotted in ; each data point is labeled with its IJK-index.
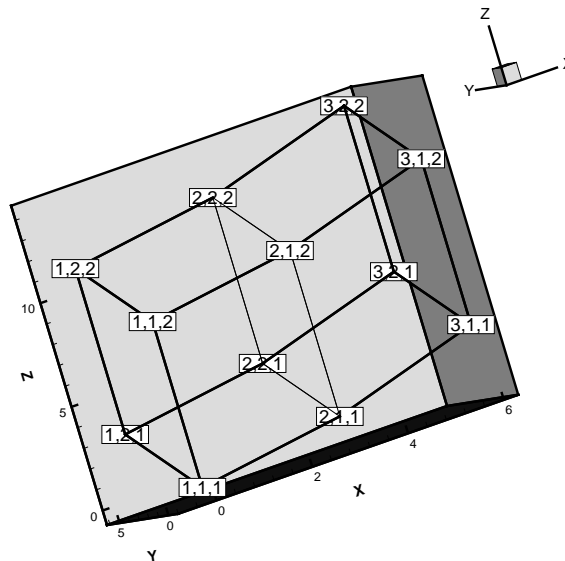

Figure 2-13. An IJK-ordered data set.

```
VARIABLES = "X" "Y" "Z" "Temp"
ZONE I=3, J=2, K=2,DATAPACKING=POINT

 0 0 0 0
```

```
3 0 1 5
6 0 3 10
0 6 3 10
3 6 4 41
6 6 6 72
0 0 8 0
3 0 9 29
6 0 11 66
0 6 11 66
3 6 12 130
6 6 14 169
```

BLOCK Format of the Same Data. The same data set as Section , this time in **BLOCK** format, is shown below. For this example, each column of data corresponds to a data point; each row to a variable.

```
VARIABLES = "X" "Y" "Z" "Temp"
ZONE I=3, J=2, K=2, DATAPACKING=BLOCK
 0 3 6 0 3 6 0 3 6 0 3 6
 0 0 0 6 6 6 0 0 0 6 6 6
 0 1 3 3 4 6 8 9 11 11 12 14
 0 5 10 10 41 72 0 29 66 66 130 169
```

FORTRAN Code to Generate an IJK-Ordered Zone in POINT Format

Example. The following sample FORTRAN code shows how to create an IJK-ordered zone in **POINT** format:

```
 WRITE (*,*) 'VARIABLES = "X", "Y", "Z", "Temp"'
 WRITE (*,*) 'ZONE I=',IMAX,' J=',JMAX,' K=',KMAX,'
DATAPACKING=POINT'
DO 1 K=1,KMAX
   DO 1 J=1,JMAX
      DO 1 I=1,IMAX
1         WRITE (*,*) X(I,J,K), Y(I,J,K), Z(I,J,K), Temp(I,J,K)
```

In **BLOCK** format, all *IMax*JMax*KMax* values of each variable are listed, one variable at a time. Within each variable block, all the values of the variable at each data point are listed.

FORTRAN Code to Generate IJK-Ordered Data in BLOCK Format Example. The following sample FORTRAN code shows how to create an IJK-ordered zone in **BLOCK** format:

```
    INTEGER VAR
      .
      .
      .
      .
    WRITE (*,*) ´ZONE DATAPACKING=BLOCK, I=´, IMAX, ´, J=´, JMAX, ´,
  K=´, KMAX
   DO 1 VAR=1,NUMVAR
        DO 1 K=1,KMAX
           DO 1 J=1,JMAX
              DO 1 I=1,IMAX
1                WRITE (*,*) ARRAY(VAR,I,J,K)
```

One Variable Data Files

For ordered data, it is possible to read in a data file that has only one variable. Tecplot then creates the other required variables. That is, if your data is I-ordered, a variable containing the I-index values is created, numbered **V1**, and called **I**. For IJ-ordered data, two variables, numbered **V1** and **V2** and called **I** and **J**, are created to contain the I- and J-index values. For IJK-ordered data, three variables **I**, **J**, and **K** are created and numbered **V1**, **V2**, and **V3**. The variable in the data file is numbered with the next available variable number, that is, **V2** for I-ordered data, **V3** for IJ-ordered data, and **V4** for IJK-ordered data. The created variables are the default X-, Y-, and Z-variables. The data type for the created variables is determined according to the following table:

Maximum of IMax, JMax, and KMax	Data Type
< 256	**BYTE**
<32,766	**SHORTINT**
>=32,766	**SINGLE**

For example, if you have an ASCII file with 256 by 384 numbers representing intensities of a rasterized image, you could make a data file similar to the following:

```
  VARIABLES = "TEMPERATURE"
   ZONE I=256, J=384
  List all 98,304 values of temperature here.
```

Read the data file into Tecplot. Two new variables of type **SHORTINT** are created and used as the default X- and Y-coordinates. These variables are the I- and J-index values; they are named **I** and **J**. You can now create any type of 2D plot with the data.

If you have finite-element data, Tecplot will not create any new variables for you. If you need to add variables to finite-element data, you can do so using the Data menu.

2 - 4 Finite-Element Data

For finite-element data, the zone types, specified in the **ZONETYPE** parameter, may be **FELINESEG**, **FETRIANGLE**, **FEQUADRILATERAL**, **FETETRAHEDRON**, or **FEBRICK**. For any of these **DATAPACKING** may be **POINT** or **BLOCK**.

The number of nodes (data points) is given by the **N=***numnodes* parameter, and the number of elements is given by the **E=***numelements* parameter (this is also the total length of the connectivity list).

Zone data is divided into two logical sections. It has no markers, but you may place blank lines between the sections to distinguish them. The first section, the node (and sometimes element) data, lists the values of the variables at the data points (or nodes) or cell-centers (elements) as if they were I-ordered (one-dimensional) zone data. The second section, the connectivity list, defines how the nodes are connected to form elements. There must be *numelements* lines in the second section; each line defines one element. The number of nodes per line in the connectivity list depends on the element type specified in the zone control line (**ZONETYPE** parameter). For example, **ZONTYPE=FETRIANGLE** has three numbers per line in the connectivity list. If nodes 5, 7, and 8 are connected, one line reads: **5 7 8**.

In the descriptions below, **N***E* is the *E*th node at a vertex of an element. The subscripts of **N***E* refer to the element number. For example, **N2**$_3$ represents the second node of the third element.

For the line segment element type, each line of the connectivity list contains two node numbers that define a linear element:

$$N1_M, N2_M$$

For the triangle element type, each line of the connectivity list contains three node numbers that define a triangular element:

$$N1_M, N2_M, N3_M$$

For the quadrilateral element type, each line of the connectivity list contains four node numbers that define a quadrilateral element:

$N1_M$, $N2_M$, $N3_M$, $N4_M$

If you need to mix quadrilateral and triangle elements, either create two zones or use the quadrilateral element type with node numbers ($N4_M$=$N3_M$) repeated to form triangles.

Zones created from the quadrilateral and triangle element types are called FE-surface zones.

For the tetrahedron element type, each line of the second section of the zone data contains four node numbers that define a tetrahedral element:

$N1_M$ $N2_M$ $N3_M$ $N4_M$

For the brick element type, each line of the second section contains eight node numbers that define a "brick-like" element:

$N1_M$ $N2_M$ $N3_M$ $N4_M$ $N5_M$ $N6_M$ $N7_M$ $N8_M$

Tecplot divides the eight nodes into two groups of four; nodes $N1_M$, $N2_M$, $N3_M$, and $N4_M$ make up the first group, and $N5_M$, $N6_M$, $N7_M$, and $N8_M$ make up the second group. Each node is connected to two nodes within its group and the node in the corresponding position in the other group. For example, $N1_M$ is connected to $N2_M$ and $N4_M$ in its own group, and to $N5_M$ in the second group. To create elements with fewer than eight nodes, repeat nodes as necessary, keeping in mind the basic brick connectivity just described. Figure 2-14 shows the basic brick connectivity. For example, to create a tetrahedron, you can set $N3_M$=$N4_M$ and $N5_M$=$N6_M$ =$N7_M$=$N8_M$. To create a quadrilateral-based pyramid, you can set $N5_M$=$N6_M$=$N7_M$=$N8_M$. If you need a mixture of bricks and tetrahedra, either use two zones or use the brick element type with node numbers repeated so that tetrahedra result.
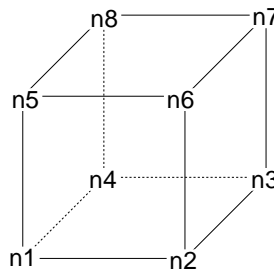

Figure 2-14. Basic brick connectivity.

Zones created from the brick and tetrahedron element types are called finite-element volume zones.

If **CONNECTIVITYSHAREZONE=***nnn* is in the zone control line, the connectivity list is shared from zone *nnn*. In this case, no connectivity list is given, just the node (and possibly element) data. If *nnn* is greater than or equal to the current zone number, Tecplot generates an error message.

Triangle Data in POINT Format Example

An example of triangle element type finite-element data with **POINT** datapacking is listed below. There are two variables (**X**, **Y**) and five data points. In this example, each row of the data section corresponds to a node and each column to a variable. Each row of the connectivity list corresponds to a triangular element and each column specifies a node number. This data set is plotted in Figure 2-15. Each data point is labeled with its node number.
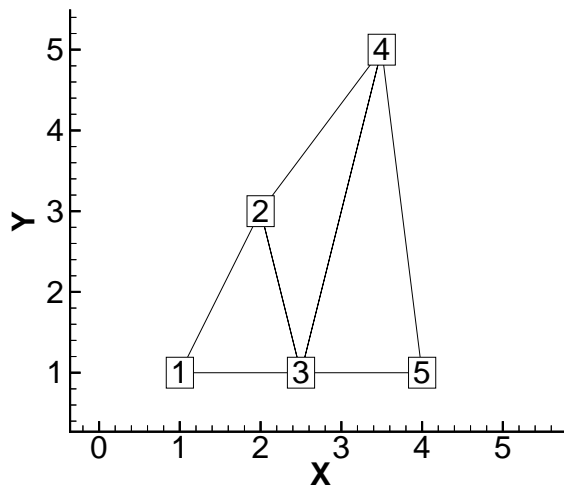
Figure 2-15. A finite-element triangle data set.

```
VARIABLES = "X", "Y"
ZONE N=5, E=3, DATAPACKING=POINT, ZONETYPE=FETRIANGLE
1.0 1.0
2.0 3.0
2.5 1.0
3.5 5.0
4.0 1.0

1 2 3
3 2 4
3 5 4
```

BLOCK Format of the Same Data. The same data in **BLOCK** format is shown below. In this example, each column of the data section corresponds to a node and each row to a variable. As above, each row of the connectivity list corresponds to a triangular element and each column specifies a node number.

```
VARIABLES = "X", "Y"
 ZONE N=5, E=3, DATAPACKING=BLOCK, ZONETYPE=FETRIANGLE
 1.0 2.0 2.5 3.5 4.0
 1.0 3.0 1.0 5.0 1.0
 1 2 3
 3 2 4
 3 5 4
```

FORTRAN Code Generating Triangle Data in POINT Format Example

The following sample FORTRAN code shows how to create triangle element type finite-element data in **POINT** format:

```
    INTEGER VAR
    .
    .
    .
    WRITE (*,*) ´ZONE DATAPACKING=POINT, ZONETYPE=FETRIANGLE,N=´,
  &    NNODES,´,E=´,NELEM
    DO 1 N=1,NNODES
      DO 1 VAR=1,NUMVAR
1        WRITE(*,*) VARRAY(VAR,N)

    DO 2 M=1,NELEM
      DO 2 L=1,3
2        WRITE (*,*) NDCNCT(M,L)
```

FORTRAN Code Generating Triangle Data in BLOCK Format Example

This FORTRAN code creates triangle element type finite-element data in **BLOCK** format:

```
    INTEGER VAR
    .
    .
    .
    WRITE (*,*) ´ZONE DATAPACKING=BLOCK,
ZONETYPE=FETRIANGLE,N=´,NNODES,
```

```
  &´,E=´,NELEM
     DO 1 VAR=1,NUMVAR
       DO 1 N=1,NNODES
1        WRITE(*,*) VARRAY(VAR,N)
     DO 2 M=1,NELEM
       DO 2 L=1,3
2        WRITE (*,*) NDCNCT(M,L)
```

Finite-Element Zone Node Variable Parameters Example

The node variable parameter allows setting of the connectivity to match the value of the selected node variable. In the example below, the files appear to be identical in Tecplot, although the connectivity list has changed to reflect the values of the node variable node order. Notice that the index value of the nodes is not changed by the node variable value.

The original data set:

```
TITLE     = "Data with original node ordering"
VARIABLES = "X"
"Y"
ZONE T="Triangulation"
 N=6, E=5,DATPACKING=POINT, ZONETYPE=FETRIANGLE
DT=(SINGLE SINGLE)
 2.00E+000 3.00E+000
 2.20E+000 3.10E+000
 3.10E+000 4.20E+000
 2.80E+000 3.50E+000
 2.40E+000 2.10E+000
 4.30E+000 3.20E+000
 1 2 5
 6 4 3
 5 4 6
 2 3 4
 5 2 4
```

The data set with the nodes re-ordered for connectivity:

```
TITLE     = "Data with modified node ordering"
VARIABLES = "X"
"Y" "Node-Order"
```

```
ZONE T="Triangulation"
 N=6, NV = 3, E=5,DATAPACKING=POINT, ZONETYPE=FETRIANGLE
DT=(SINGLE SINGLE)
 2.00E+000 3.00E+000 5
 2.20E+000 3.10E+000 4
 3.10E+000 4.20E+000 1
 2.80E+000 3.50E+000 2
 2.40E+000 2.10E+000 6
 4.30E+000 3.20E+000 3
 5 4 6
 3 2 1
 6 2 3
 4 1 2
 6 4 2
```

2- 4.1 Variable and Connectivity List Sharing

The **VARSHARELIST** in the **ZONE** record allows you to share variables from specified previous zones. The **CONNECTIVITYSHAREZONE** parameter in the **ZONE** record allows you to share the connectivity list from a specified previous zone. The following is an example to illustrate these features.

The table below shows Cartesian coordinates X and Y of six locations, and the pressure measured there at three different times (*P1*, *P2*, *P3*). The XY locations have been arranged into finite-elements.

X	Y	P_1	P_2	P_3
-1.0	0.0	100	110	120
0.0	0.0	125	135	145
1.0	0.0	150	160	180
-0.5	0.8	150	165	175
0.5	0.8	175	185	195
0.0	1.6	200	200	200

For this case, we want to set up three zones in the data file, one for each time measurement. Each zone has three variables: X, Y, and P. The zones are of the triangle element type, meaning that three nodes must be used to define each element. One way to set up this data file would be to list the

complete set of values for X, Y, and P for each zone. Since the XY-coordinates are exactly the same for all three zones, a more compact data file can be made by using the **VARSHARELIST**. In the data file given below, the second and third zones have variable sharing lists that share the values of the X- and Y-variables and the connectivity list from the first zone. As a result, the only values listed for the second and third zones are the pressure variable values. Note that the data could easily have been organized in a single zone with five variables. Since blank lines are ignored in the data file, you can embed them to improve readability. A plot of the data is shown in .
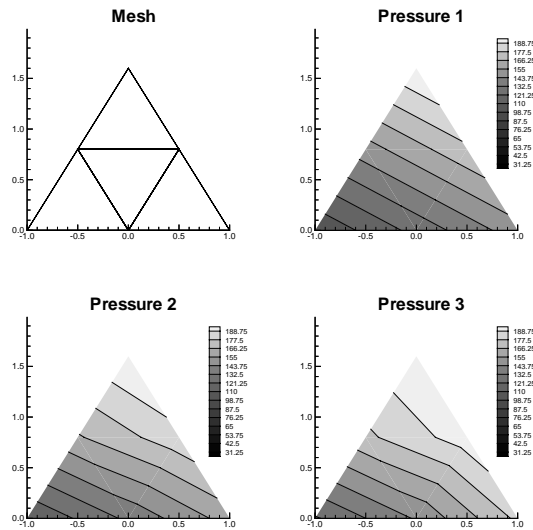
Figure 2-16. A plot of finite-element zones.

```
TITLE = "Example: Variable and Connectivity List Sharing"
VARIABLES = "X", "Y", "P"
ZONE T="P_1", DATAPACKING=POINT, N=6, E=4, ZONETYPE=FETRIANGLE
-1.0 0.0 100
0.0 0.0 125
1.0 0.0 150
-0.5 0.8 150
0.5 0.8 175
0.0 1.6 200

1 2 4
```

```
2 5 4
3 5 2
5 6 4
ZONE T="P_2", DATAPACKING=POINT, N=6, E=4, ZONETYPE=FETRIANGLE,
VARSHARELIST = ([1, 2]=1), CONNECTIVITYSHAREZONE = 1
110 135 160 165 185 200


ZONE T="P_3", DATAPACKING=POINT, N=6, E=4, ZONETYPE=FETRIANGLE,
VARSHARELIST = ([1, 2]=1), CONNECTIVITYSHAREZONE = 1
120 145 180 175 195 200
```

2- 4.2 ASCII Data File Conversion to Binary

Although Tecplot can read and write ASCII or binary data files, binary data files are more compact and are read into Tecplot much more quickly. Your Tecplot distribution includes Preplot, which converts ASCII to binary data files. You can also use Preplot to debug ASCII data files Tecplot cannot read.

Preplot Options

To use Preplot, type the following command from the UNIX shell prompt, from a DOS prompt, or using the Run command in Windows:

```
preplot infile [outfile] [options]
```

where *infile* is the name of the ASCII data file, *outfile* is an optional name for the binary data file created by Preplot, and *options* is a set of options from either the standard set of Preplot options or from a special set of options for reading PLOT3D format files. If *outfile* is not specified, the binary data file has the same base name as the *infile* with a **.plt** extension. You may use a minus sign ("-") in place of either the *infile* or *outfile* to specify standard input or standard output, respectively.

Any or all of **-iset**, **-jset**, and **-kset** can be set for each zone, but only one of each per zone.

For more Preplot command lines, see Appendix B.3, "Preplot."

Preplot Examples

If you have an ASCII file named **dset.dat**, you can create a binary data file called **dset.plt** with the following Preplot command:

```
preplot dset.dat dset.plt
```

By default, Preplot looks for files with the **`.dat`** extension, and creates binary files with the **`.plt`** extension. Thus, either of the following commands is equivalent to the above command:

```
preplot dset
preplot dset.dat
```

Preplot checks the input ASCII data file for errors such as illegal format, numbers too small or too large, the wrong number of values in a data block, and illegal finite-element node numbers. If Preplot finds an error, it issues a message displaying the line and column where the error was first noticed. This is only an indication of where the error was *detected*; the actual error may be in the preceding columns or lines.

If Preplot encounters an error, you may want to set the debug option to get more information about the events leading up to the error:

```
preplot dset.dat -d
```

You can set the flag to **`-d2`**, or **`-d3`**, or **`-d4`**, and so forth, to obtain even more detailed information.

In the following Preplot command line, the number of points that are written to the binary data file **`dset.plt`** is less than the number of points in the input file **`dset.dat`**:

```
preplot dset.dat -iset 3,6,34,2 -jset 3,1,21,1 -iset 4,4,44,5
```

For zone 3, Preplot outputs data points with I-index starting at 6 and ending at 34, skipping every other one, and J-index starting at one and ending at 21. For zone 4, Preplot outputs data points with the I-index starting at four, ending at 44, and skipping by five.

In the following Preplot command line, every other point in the I-, J-, and K-directions is written to the binary data file:

```
preplot dset.dat -iset ,,,2 -jset ,,,2 -kset ,,,2
```

The *zone*, *start*, and *end* parameters are not specified, so all zones are used, starting with index 1, and ending with the maximum index. The overall effect is to reduce the number of a data points by a factor of about eight.

Preplot Conversion of the PLOT3D Format

PLOT3D is a graphics plotting package developed at NASA. Some numerical simulation packages and other programs can create graphics in PLOT3D format. There are two paths by which you can get files in PLOT3D format into Tecplot. This section describes the Preplot path; you can also use

the PLOT3D Loader described in Section E - 13 "PLOT3D Data Loader" on page 654 in the Tecplot 360 User's Manual Plot3D Data Loader in the Tecplot 360 User's Manual. The PLOT3D Loader has more advanced capabilities than Preplot.

Preplot can read files in the PLOT3D format and convert them to Tecplot binary data files through the use of special switches. You do not need to know about these switches unless you have data in PLOT3D format.

PLOT3D files typically come in pairs consisting of a grid file (with extension `.g`) and a solution file (with extension `.q`). Sometimes only the grid file is available. The grid itself may be either a single grid, or a multigrid, and the data may be 1D, 2D, 3D-planar, or 3D-whole (equivalent to Tecplot's 3D volume data). The PLOT3D files may be binary or ASCII. The PLOT3D-specific switches to Preplot allow you to read PLOT3D files with virtually any combination of these options.

The *ilist*, *jlist*, and *klist* are comma-separated lists of items of the form:

> `start[:end][:skip]]`

where *start* is the number of the starting I-, J-, or K-plane, *end* is the number of ending I-, J-, or K-plane, and *skip* is the skip factor between planes. If *end* is omitted, it defaults to the starting plane (so if just *start* is specified, only that one plane is included). The *skip* defaults to one (every plane) if omitted; a value of two includes every other plane, a value of three include every third plane, and so on.

You must specify one of the flags `-1d`, `-2d`, `-3dp`, or `-3dw`. You may also specify only one of `-ip`, `-jp`, or `-kp` and only one of `-b` or `-f`.

If the input PLOT3D file is 3D whole (`-3dw`) and none of the plane-extraction switches `-ip`, `-jp`, or `-kp` is specified, the PLOT3D file is converted directly to an IJK-ordered zone (or multiple zones if the file is multi-grid).

For example, in the following command line, Preplot reads from the PLOT3D files `aero.g` and `aero.q`. The input is binary and 3D whole. The J-planes 2, 3, 4, 45, 46, and 47 are processed and made into six IJ-ordered zones, in a binary data file named `aero.plt`:

> `preplot aero -plot3d -b -3dw -jp 2,3,4,45,46,47`

In the following command line, the plane-extraction switches are omitted, so Preplot creates a single IJK-ordered zone:

> `preplot aero -plot3d -b -3dw`

The following command line reads an ASCII file **airplane.g** for which there is no corresponding **.q** file; the data is 3D whole:

```
preplot airplane -plot3d -gridonly -3dw
```

The following command line reads a multi-grid, 3D planar, binary-FORTRAN pair of PLOT3D files, **multgrid.g** and **multgrid.q**:

```
preplot multgrid -plot3d -m -f -3dp
```

2- 4.3 Finite-Element Data Sets

Creating a finite-element data set is generally more complicated than creating a similar-sized ordered data set[1]. In addition to specifying all the data points, you must also specify the connectivity list. Consider the data shown in Table 2 - 1.

Node	X	Y	P	T
A	0.0	1.0	100.0	1.6
B	1.0	1.0	150.0	1.5
C	3.0	1.0	300.0	2.0
D	0.0	0.0	50.0	1.0
E	1.0	0.0	100.0	1.4
F	3.0	0.0	200.0	2.2
G	4.0	0.0	400.0	3.0
H	2.0	2.0	280.0	1.9

Table 2 - 1: Finite Element Data

You can create a **POINT** Tecplot data file for this data set as follows (a 2D mesh plot of this data set is shown in Figure 2-17):

```
TITLE = "Example: 2D Finite-Element Data"
VARIABLES = "X", "Y", "P", "T"
ZONE N=8, E=4, DATAPACKING=POINT, ZONETYPE=FEQUADRILATERAL
0.0 1.0 100.0 1.6
1.0 1.0 150.0 1.5
3.0 1.0 300.0 2.0
```

---

1. Background information for FE data sets is provided in 2 - 2 "Finite-Element Data" on page 51 in the Tecplot 360 User's Manual the Tecplot 360 User's Manual - Finite Element Data.

```
0.0 0.0 50.0 1.0
1.0 0.0 100.0 1.4
3.0 0.0 200.0 2.2
4.0 0.0 400.0 3.0
2.0 2.0 280.0 1.9
1 2 5 4
2 3 6 5
6 7 3 3
3 2 8 8
```
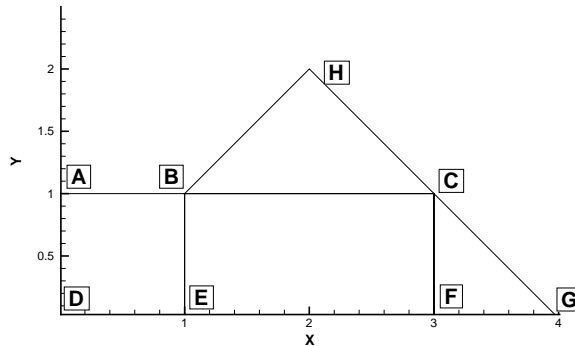

Figure 2-17. A mesh plot of 2D finite-element data.

The **ZONE** record describes completely the form and format of the data set: there are eight nodes, indicated by the parameter **N=8**; four elements, indicated by the parameter **E=4**, and the elements are all quadrilaterals, as indicated by the parameter **ZONETYPE=FEQUADRILATERAL**.

The same data file can be written more compactly in **BLOCK** format as follows:

```
TITLE = "Example: 2D Finite-Element Data"
VARIABLES = "X", "Y", "P", "T"
ZONE N=8, E=4, DATAPACKING=BLOCK, ZONETYPE=FEQUADRILATERAL
0.0 1.0 3.0 0.0 1.0 3.0 4.0 2.0
1.0 1.0 1.0 0.0 0.0 0.0 0.0 2.0
100.0 150.0 300.0 50.0 100.0 200.0 400.0 280.0
1.6 1.5 2.0 1.0 1.4 2.2 3.0 1.9
1 2 5 4
2 3 6 5
6 7 3 3
3 2 8 8
```

In **BLOCK** format, all values for a single variable are written in a single block. The length of the block is the number of data points in the zone. In **POINT** format, all variables for a single data point are written in a block, with the length of the block equal to the number of variables.

> The connectivity list is the same for both **POINT** and **BLOCK** formats.

You can change the connectivity list to obtain a different mesh for the same data points. In the above example, substituting the following connectivity list yields the five-element mesh shown in Figure 2-18. (You must also change the **E** parameter in the zone control line to specify five elements.)
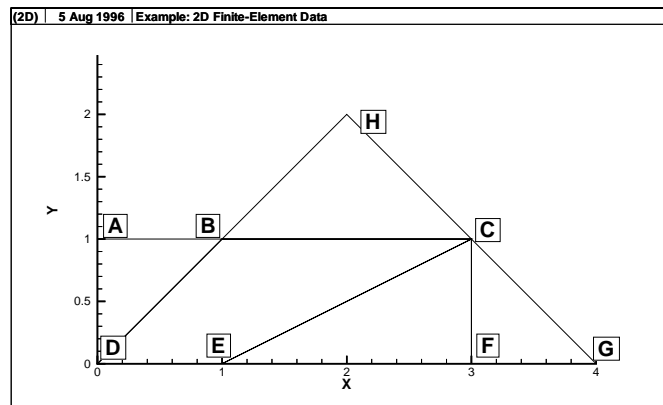
Figure 2-18. Finite-element data of Figure 2-17 with different connectivity list

```
1 2 4 4
4 2 3 5
5 3 6 6
6 7 3 3
3 2 8 8
```

FE surface data

Finite-element surface data specify node locations in three dimensions. Consider the data in Table 2-1. Locations are listed for eleven nodes, each having only the three spatial variables X, Y, and Z. We would like to create an finite-element surface zone with this data set, where some of the ele-

ments are triangles and some are quadrilaterals. All the elements could be organized into one zone, of element type Quadrilateral, but as an illustration of creating 3D surface data, create three zones: one triangular, one quadrilateral, and one a mixture (using quadrilaterals with repeated nodes for the triangles).

X	Y	Z
0.0	0.0	1.0
0.0	0.0	-2.0
1.0	0.0	-2.0
1.0	1.0	0.0
1.0	1.0	-1.0
1.0	-1.0	0.0
1.0	-1.0	-1.0
-1.0	1.0	0.0
-1.0	1.0	-1.0
-1.0	-1.0	0.0
-1.0	-1.0	-1.0

Table 2-1. Data set with eleven nodes and three variables.

A Tecplot data file for the data in is shown below in **POINT** format and plotted in :

```
TITLE = "Example: 3D FE-SURFACE ZONES"
VARIABLES = "X", "Y", "Z"
ZONE T="TRIANGLES", N=5, E=4, DATAPACKING=POINT, ZONETYPE=FETRIANGLE
0.0 0.0 1.0
-1.0 -1.0 0.0
-1.0 1.0 0.0
1.0 1.0 0.0
1.0 -1.0 0.0
1 2 3
1 3 4
1 4 5
1 5 2
ZONE T="PURE-QUADS", N=8, E=4, DATAPACKING=POINT,
                 ZONETYPE=FEQUADRILATERAL
-1.0 -1.0 0.0
```

```
-1.0 1.0 0.0
1.0 1.0 0.0
1.0 -1.0 0.0
-1.0 -1.0 -1.0
-1.0 1.0 -1.0
1.0 1.0 -1.0
1.0 -1.0 -1.0
1 5 6 2
2 6 7 3
3 7 8 4
4 8 5 1
ZONE T="MIXED", N=6, E=4, DATAPACKING=POINT,
ZONETYPE=FEQUADRILATERAL
-1.0 -1.0 -1.0
-1.0 1.0 -1.0
1.0 1.0 -1.0
1.0 -1.0 -1.0
0.0 0.0 -2.0
1.0 0.0 -2.0
1 5 2 2
2 5 6 3
3 4 6 6
4 1 5 6
```
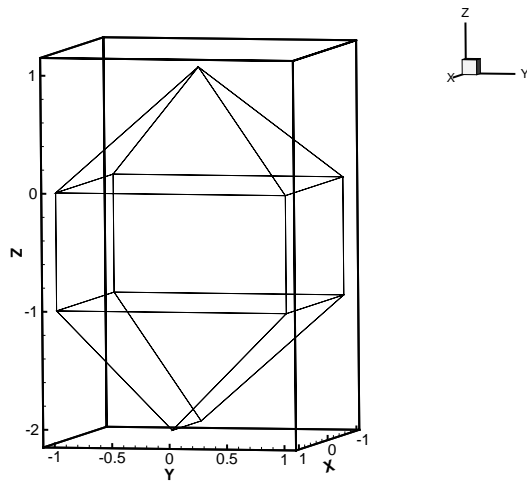

Figure 2-19. Three-dimensional mesh plot of finite-element surface data.

FE Volume Data Files

Finite-element volume data in Tecplot is constructed from either tetrahedrons having four nodes or bricks having eight nodes. Bricks are more flexible, because they can be used (through the use of repeated nodes in the connectivity list) to construct elements with fewer than eight nodes and combine those elements with bricks in a single zone. Tetrahedrons, on the other hand, are harder to construct because care must be taken to ensure all faces in all of the tetrahedron are planar in physical space.

Finite-Element Volume - Brick Data Set

As a simple example of finite-element volume brick data, consider the data in Table 2 - 2. The data can be divided into five brick elements, each of which is defined by eight nodes.

X	Y	Z	Temperature
0.0	0.0	0.0	9.5
1.0	1.0	0.0	14.5
1.0	0.0	0.0	15.0
1.0	1.0	1.0	16.0
1.0	0.0	1.0	15.5
2.0	2.0	0.0	17.0
2.0	1.0	0.0	17.0
2.0	0.0	0.0	17.5
2.0	2.0	1.0	18.5
2.0	1.0	1.0	20.0
2.0	0.0	1.0	17.5
2.0	2.0	2.0	18.0
2.0	1.0	2.0	17.5
2.0	0.0	2.0	16.5

Table 2 - 2: Finite-Element Volume - Brick Data Set. Data with 14 nodes and four variables.

In each element's connectivity list, Tecplot draws connections from each node to three other nodes. You can think of the first four nodes in the element as the "bottom" layer of the brick, and the second four nodes as the "top." Within the bottom or top layer, nodes are connected cyclically (1-2-3-4-1; 5-6-7-8-5); the layers are connected by connecting corresponding nodes (1-5; 2-6; 3-7; 4-8). Figure 2-14 illustrates this basic connectivity. When you are creating your own connectivity lists for brick elements, you must keep this basic connectivity in mind, particularly when using dupli-

cated nodes to create pyramids and wedges. Tecplot lets you create elements that violate this basic connectivity, but the result will probably not be what you want.

The data file in **POINT** format is shown below:

```
TITLE = "Example: FE-Volume Brick Data"
VARIABLES = "X", "Y", "Z", "Temperature"
ZONE N=14, E=5, DATAPACKING=POINT, ZONETYPE=BRICK
0.0 0.0 0.0 9.5
1.0 1.0 0.0 14.5
1.0 0.0 0.0 15.0
1.0 1.0 1.0 16.0
1.0 0.0 1.0 15.5
2.0 2.0 0.0 17.0
2.0 1.0 0.0 17.0
2.0 0.0 0.0 17.5
2.0 2.0 1.0 18.5
2.0 1.0 1.0 20.0
2.0 0.0 1.0 17.5
2.0 2.0 2.0 18.0
2.0 1.0 2.0 17.5
2.0 0.0 2.0 16.5

1 1 1 1 2 4 5 3
2 4 5 3 7 10 11 8
4 4 5 5 10 13 14 11
4 4 4 4 9 12 13 10
2 2 4 4 7 6 9 10
```

The same data in **BLOCK** format is shown below:

```
TITLE = "Example: FE-Volume Brick Data"
VARIABLES = "X", "Y", "Z", "Temperature"
ZONE N=14, E=5, DATAPACKING=BLOCK, ZONETYPE=FEBRICK
0.0 1.0 1.0 1.0 1.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
0.0 1.0 0.0 1.0 0.0 2.0 1.0 0.0 2.0 1.0 0.0 2.0 1.0 0.0
0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 2.0 2.0 2.0
9.5 14.5 15.0 16.0 15.5 17.0 17.0
17.5 18.5 20.0 17.5 18.0 17.5 16.5

1 1 1 1 2 4 5 3
2 4 5 3 7 10 11 8
```

```
4 4 5 5 10 13 14 11
4 4 4 4 9 12 13 10
2 2 4 4 7 6 9 10
```

Figure 2-20 shows the resulting mesh plot from the data set listed in this section.
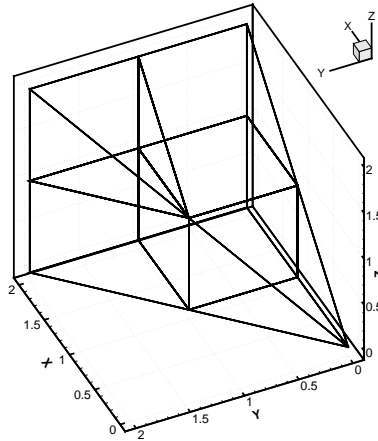
Figure 2-20. A finite-element brick zone.

Finite-Element Volume - Tetrahedral Data Set

As a simple example of an finite-element volume data set using tetrahedral elements, consider the data in Table 2 - 3. The data set consists of thirteen nodes, with seven variables. The nodes are to be connected to form twenty tetrahedral elements, each with four nodes.

X	Y	Z	C	U	V	W
0	0	-95	-1	1	0	8
0	85	-42	0	-5	-3	9
81	26	-42	2	-22	80	8
50	-69	-42	-6	72	52	9
-50	-69	-42	14	67	-48	9
-81	26	-2	20	-30	-82	9
0	0	0	1	-2	-5	10
50	69	43	14	-68	48	11

Table 2 - 3: Finite-Element Volume - Tetrahedral data set with 13 nodes and seven variables.

X	Y	Z	C	U	V	W
81	-26	43	20	31	82	11
0	-85	43	0	84	-3	10
-81	-26	43	2	21	-80	11
-50	69	43	-6	-71	-51	11
0	0	96	1	0	-1	12

Table 2 - 3: Finite-Element Volume - Tetrahedral data set with 13 nodes and seven variables.

The data file in **POINT** format for the data in Table 2 - 3 is shown below, and plotted in Figure 2-21:

```
TITLE = "Example: FE-Volume Tetrahedral Data"
VARIABLES = "X", "Y", "Z", "C", "U", "V", "W"
ZONE N=13, E=20, DATAPACKING=POINT, ZONETYPE=FETETRAHEDRON
0 0 -95 -1 1 0 8
0 85 -42 0 -85 -3 9
81 26 -42 2 -22 80 8
50 -69 -42 -6 72 52 9
-50 -69 -42 14 67 -48 9
-81 26 -42 20 -30 -82 9
0  0 0 1 -2 -5 10
50 69 43 14 -68 48 11
81 -26 43 20 31 82 11
0 -85 43 0 84 3 10
-81 -26 43 2 21 -80 11
-50 69 43 -6 -71 -51 11
0  0 96 1 0 -1 12
1 2 3 7
1 3 4 7
1 4 5 7
1 5 6 7
1 6 2 7
2 8 3 7
3 9 4 7
4 10 5 7
5 11 6 7
6 12 2 7
12 2 8 7
8 3 9 7
9 4 10 7
```

```
10 5 11 7
11 6 12 7
12 8 13 7
8 9 13 7
9 10 13 7
10 11 13 7
11 12 13 7
```
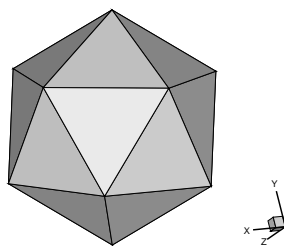

Figure 2-21. Finite-element volume
tetrahedral data.

This data file is included in your Tecplot distribution's **examples/data** directory as the file **fetetpt.dat**. A block format version of the same data is included as the file **fetetbk.dat.**

Chapter 3   ***Binary Data***

This chapter is intended only for advanced users of Tecplot who have a solid background in UNIX or Windows and application programming. Support for topics discussed in this chapter may be limited. Regular technical support is not intended to help you program your application to use the direct data file capabilities of Tecplot.

Data files for Tecplot are commonly created as output from an application program. These files are most often in ASCII format, and are then converted to a binary format with Preplot.

Included with your distribution of Tecplot is a library that contains utility functions that you can link with your application program to create binary data files directly, bypassing the use of ASCII files. This allows for fewer files to manage, conserves on disk space, and saves the extra time required to convert the files.

In UNIX, the utility functions discussed below are available in the library archive **tecio.a** which is located in the **lib** sub-directory of the Tecplot Home Directory. Under Windows, this library is called **TecIO.dll** and is located in the **bin** sub-directory. Instructions on compiling and linking using the **TECIO** library can be found in the **readme.doc** file in the **util/tecio** sub-directory under the **TECHOME** directory.

Tecplot 360 introduces a new set of TECIO functions to take full advantage of the new capabilities it offers. Each of these functions has a suffix of "110" to differentiate it from previous editions. Please note that all previously existing TECIO functions still exist and are supported for backward compatibility.

3 - 1 Function Summary

The following functions are available from the TECIO archive. For historical reasons, these functions have a FORTRAN flavor to them, both in how they are named and the way in which the parameters are passed.

Tecplot 360 TECIO Functions:

- TECINI110: Initialize the process of writing a binary data file.

- TECZNE110: Write information about the next zone to be added to the data file.

- TECFOREIGN110: Write the binary file in foreign byte order.

- TECDAT110: Write an array of data to the data file.

- TECNOD110: Write an array of node data to the data file.

- TECLAB110: Write a custom label record to the data file.

- TECGEO110: Write a geometry record to the data file.

- TECTXT110: Write a text record to the data file.

- TECFIL110: Switch output context to a different file.

- TECEND110: Close the data file.

- TECUSR110: Write a character string to the data file in a USERREC record.

- TECAUXSTR110: Write auxiliary data for the data set to the data file.

- TECZAUXSTR110:Write auxiliary data for the current zone to the data file.

- TECVAUXASTR110:Write auxiliary data for a variable to the data file.

- TECFACE110: Write the face connections for the current zone to the data file.

3 - 2 Deprecated Binary Functions

The following functions have been deprecated. We recommend replacing these function calls with the equivalent Tecplot 360 TecIO function. The deprecated functions will continue to work, but may lack the full functionality of the new functions.

```
TECFOREIGN100
TECINI100
TECZNE100
TECDAT100
TECNOD100
TECEND100
TECLAB100
TECUSR100
```

```
TECGEO100
TECTXT100
TECFIL100
TECAUXSTR100
TECZAUXSTR100
TECVAUXSTR100
TECFACE100

TECINI
TECZNE
TECDAT
TECNOD
TECEND
TECLAB
TECUSR
TECGEO
TECTXT
TECFIL
```

3 - 3 Binary Data File Function Calling Sequence

Multiple data files can be written to at the same time. For a given file, the binary data file functions must be called in a specific order.

The correct order is as follows:

```
TECFOREIGN110 (optional)
TECINI110
 TECAUXSTR110
 TECVAUXSTR110
 TECZNE110    (One or more to create multiple zones)
    TECDAT110 (One or more to fill each zone)
    TECNOD110 (One for each finite element zone)
    TECFACE110 (One for each zone with face connections)
    TECZAUXSTR110
 TECLAB110
 TECGEO110
 TECTXT110
```

```
    TECUSR110
  TECEND
```

Section explains how you can use the **TECFIL110** function along with the above functions to write to multiple files at the same time.

TECFOREIGN110 should be called prior to calling **TECINI110**. The **TECZNE110, TECLAB110**, **TECGEO110, TECAUXSTR110, TECVAUXSTR110** and **TECTXT110** functions can be called anywhere between the **TECINI110** and **TECEND110** functions. **TECDAT110** and **TECNOD110** (for finite-element data only) must be called immediately after the **TECZNE110** function call. **TECFACE110** (where face connections were indicated in the call to **TECZNE110)** must be called immediately after **TECNOD110** (for finite-element data) or **TECZNE110** (for ordered data). **TECZAUXSTR110** must be called following the **TECZNE110** call for the zone with which the auxiliary data is associated.

3 - 4 Writing to Multiple Binary Data Files

Each time **TECINI110** is called it sets up a new file "context." For each file context you must maintain the order of the calls as described in the previous section. The **TECFIL110** function is used to switch between file contexts. Up to 10 files can be written to at a time. **TECFIL110** can be called almost anywhere after **TECINI110** has been called. The only parameter to **TECFIL110**, an integer, n, shifts the file context to the nth open file where the files are numbered relative to the order of the calls to **TECINI110**.

3 - 5 Character Strings in FORTRAN

All character string parameters in FORTRAN must terminate with a null character. This is done by concatenating **char(0)** to the end of a character string.

For example, to send the character string "Hi Mom" to a function called **A**, the syntax would be:

```
  I=A("Hi Mom"//char(0))
```

3 - 6 Boolean Flags

Integer parameters identified as "flags" indicate boolean values. Pass 1 for true, and 0 for false.

3 - 7 Binary Data File Function Reference

This section describes each of the **TECIO** functions in detail.

TECAUXSTR110

Summary: Writes auxiliary data for the data set to the data file. The function may be called any time between **TECINI110** and **TECEND110**. Auxiliary data may be used by text, macros, equations (if it is numeric) and add-ons. It may be viewed directly in the AuxData page of the Data Set Information dialog.

FORTRAN Syntax:

```
      INTEGER*4 FUNCTION TECAUXSTR110(Name,
  &                                    Value)
       CHARACTER*(*)  Name
       CHARACTER*(*)  Value
```

C Syntax:
```
#include TECIO.h

INTEGER4 TECAUXSTR110(char *Name,
                      char *Value)
```

Return Value: **0** if successful, **-1** if unsuccessful.

Parameters: *Name* - The name of the auxiliary data. If this duplicates an existing name, the value will overwrite the existing value. Must be a null-terminated character string.
Value - The value to assign to the named auxiliary data. Must be a null-terminated character string.

TECDAT110

Summary: Writes an array of data to the data file.Data should not be passed for variables that have been indicated as passive or shared (via **TECZNE110**).
The following table describes the order the data must be supplied given different zone types (IsBlock is a parameter supplied to **TECZNE110**):

Zone Type	Variable Location	IsBlock	Number of Values Supplied	Order
Ordered	Nodal	1	IMax* JMax* KMax* NumVars	I varies fastest, then J, then K, then V
Ordered	Nodal	0	IMax* JMax* KMax* NumVars	V varies fastest, then I, then J, then K
Ordered	Cell Centered	1	(IMax-1)* (JMax-1)* (KMax-1)* NumVars	I varies fastest, then J, then K, then V
Ordered	Cell Centered	0	Not allowed	
Finite Element	Nodal	1	IMax (i.e. NumPts * NumVars)	N varies fastest, then V
Finite Element	Nodal	0	IMax (i.e. NumPts * NumVars)	V varies fastest, then N
Finite Element	Cell Centered	1	JMax (i.e. NumElements * NumVars)	E varies fastest, then V
Finite Element	Cell Centered	0	Not allowed	

Note that if any variables are cell centered then the data must be supplied in block format thus the IsBlock parameter in TECZNE110 MUST be set to 1

TECDAT110 allows you to write your data in a piecemeal fashion in case it is not contained in one contiguous block in your program. Enough calls to **TECDAT110** must be made that the correct number of values are written for each zone and that the aggregate order for the data is correct.

In the above summary, *NumVars* is based on the number of variable names supplied in a previous call to **TECINI110**.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECDAT110(N,
```

```
             &                        Data,
             &                        IsDouble)
              INTEGER*4 N
              REAL or DOUBLE PRECISION Data(1)
              INTEGER*4 IsDouble
```

C Syntax: `#include TECIO.h`

```
INTEGER4 TECDAT110(INTEGER4 *N,
            void *Data,
            INTEGER4 *IsDouble);
```

Return Value: 0 if successful, **-1** if unsuccessful.

Parameters: *N* - Pointer to an integer value specifying number of values to write.
Data - Array of single or double precision data values.
IsDouble - Pointer to the integer flag stating whether the array Data is single (**0**)
or double (**1**) precision.

TECEND110

Summary: *Must* be called to close out the current data file. There must be a corresponding
TECEND110 for each **TECINI110**.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECEND110()
```

C Syntax: `#include TECIO.h`

```
INTEGER4 TECEND110();
```

Return Value: **0** if successful, **-1** if unsuccessful.

Parameters: None.

TECFACE110

Summary: Writes face connections for the current zone to the file. This function must be
called after **TECNOD110**, and may only be called if a non-zero value of *Num-FaceConnections* was used in the previous call to **TECZNE110**.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECFACE110(FaceConnections)
```

INTEGER*4 *FACECONNECTIONS*

C Syntax: **#include TECIO.h**

   INTEGER4 TECFACE110(INTEGER4 **FaceConnections***);**

Return Value: **0** if successful, **-1** if unsuccessful.

Parameters: *FaceConnections*

The array that specifies the face connections. The array must be dimensioned **(L, NumFace-Connections)**, where L is determined by the type of face connection specified by the **Face-NeighborMode** parameter to **TECZNE110**:

FaceNeighbor Mode	# Values	Data
LocalOneToOne	3	cz,fz,cz
LocalOneToMany	nz+4	cz,fz,oz,nz,cz1,cz2,...,czn
GlobalOneToOne	4	cz,fz,ZZ,CZ
GlobalOneToMany	2*nz+4	cz,fz,oz,nz,ZZ1,CZ1,ZZ2,CZ2,...,ZZn, CZn

Where:

cz = cell in current zone

fz = face of cell in current zone

oz = face obscuration flag (only applies to one-to-many):

  0 = face partially obscured

  1 = face entirely obscured

nz = number of cell or zone/cell associations (only applies to one-to-many)

ZZ = remote Zone

CZ = cell in remote zone

cz,fz combinations must be unique. Additionally, Tecplot assumes that with the one-to-one face neighbor modes a supplied cell face is entirely obscured by its neighbor. With one-to-many, the obscuration flag must be supplied. Faces that

are not supplied with neighbors are run through Tecplot's auto face neighbor generator (FE only).

TECFIL110

Summary: Switch output context to a different file. Each time **TECINI110** is called, a new file "context" is switched to. This allows you to write multiple data files at the same time.

FORTRAN Syntax：

```
INTEGER*4 FUNCTION TECFIL110(F)

INTEGER*4 F
```

C Syntax:

```
#include TECIO.h

INTEGER4 TECFIL110(INTEGER4 *F);
```

Return Value: **0** if successful, **-1** if unsuccessful.

Parameters: *F* - Pointer to integer specifying file number to switch to. A value of 1 indicates a switch to the file opened by the first call to **TECINI110**.

TECFOREIGN110

Summary: Sets the byte ordering request for subsequent calls to **TECINI110**. The byte ordering request will remain in effect until the next call to this function. This has no effect on files already opened via **TECINI110**, it only affects files opened by future **TECINI110** calls. This function is not required. The default is to write out native byte order.

FORTRAN Syntax：

```
INTEGER*4 FUNCTION TECFIL110(DoForeignByteOrder)

INTEGER*4 DoForeignByteOrder
```

C Syntax:

```
#include TECIO.h

INTEGER4 TECFIL110(INTEGER4 *DoForeignByteOrder);
```

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *DoForeignByteOrder* - Pointer to boolean value indicating if future files created by **TECINI110** should be written out in foreign byte order. 0 indicates native byte order. 1 indicates foreign byte order.

<div align="right">

TECGEO110

</div>

Summary: Writes a geometry to the data file.

FORTRAN Syntax:

```
      INTEGER*4 FUNCTION TECGEO110(XOrThetaPos,
     &                             YOrRPos,
     &                             ZPos,
     &                             PosCoordMode,
     &                             AttachToZone,
     &                             Zone,
     &                             Color,
     &                             FillColor,
     &                             IsFilled,
     &                             GeomType,
     &                             LinePattern,
     &                             PatternLength,
     &                             LineThickness,
     &                             NumEllipsePts,
     &                             ArrowheadStyle,
     &                             ArrowheadAttachment,
     &                             ArrowheadSize,
     &                             ArrowheadAngle,
     &                             Scope,
     &                             Clipping,
     &                             NumSegments,
     &                             NumSegPts,
     &                             XOrThetaGeomData,
     &                             YOrRGeomData,
     &                             ZGeomData,
     &                             MFC)
       DOUBLE PRECISION XOrThetaPos
```

```
        DOUBLE PRECISION YOrRPos
        DOUBLE PRECISION ZPos
        INTEGER*4 PosCoordMode
        INTEGER*4 AttachToZone
        INTEGER*4 Zone
        INTEGER*4 Color
        INTEGER*4 FillColor
        INTEGER*4 IsFilled
        INTEGER*4 GeomType
        INTEGER*4 LinePattern
        DOUBLE PRECISION PatternLength
        DOUBLE PRECISION LineThickness
        INTEGER*4 NumEllipsePts
        INTEGER*4 ArrowheadStyle
        INTEGER*4 ArrowheadAttachment
        DOUBLE PRECISION ArrowheadSize
        DOUBLE PRECISION ArrowheadAngle
      INTEGER*4 Scope
      INTEGER*4 Clipping
        INTEGER*4 NumSegments
        INTEGER*4 NumSegPts
        REAL*4 XOrThetaGeomData
        REAL*4 YOrRGeomData
        REAL*4 ZGeomData
      CHARACTER*(*) MFC
```

C Syntax: `#include TECIO.h`

```
INTEGER4 TECGEO110(double *XOrThetaPos,
                   double *YOrRPos,
                   double *ZPos,
                   INTEGER4 *PosCoordMode,
                   INTEGER4 *AttachToZone,
                   INTEGER4 *Zone,
                   INTEGER4 *Color,
                   INTEGER4 *FillColor,
                   INTEGER4 *IsFilled,
                   INTEGER4 *GeomType,
                   INTEGER4 *LinePattern,
                   double *PatternLength,
                   double *LineThickness,
```

```
                                    INTEGER4 *NumEllipsePts,
                                    INTEGER4 *ArrowheadStyle,
                                    INTEGER4 *ArrowheadAttachment,
                                    double *ArrowheadSize,
                                    double *ArrowheadAngle,
                                    INTEGER4 *Scope,
                                    INTEGER4 *Clipping,
                                    INTEGER4 *NumSegments,
                                    INTEGER4 *NumSegPts,
                                    float *XOrThetaGeomData,
                                    float *YOrRGeomData,
                                    float *ZGeomData,
                                  char *MFC)
```

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *XOrThetaPos* - Pointer to double value specifying the X-position or, for polar line plots, the Theta-position of the geometry.
YOrRPos - Pointer to double value specifying the Y-position or, for polar line plots, the R-position of the geometry.
ZPos - Pointer to double value specifying the Z-position of the geometry.
PosCoordMode - Pointer to integer value specifying the position coordinate system.

> **0=Grid**
> **1=Frame**
> **6=Grid3D**

AttachToZone - Pointer to integer flag to signal that the geometry is "attached" to a zone.
Zone - Pointer to integer value specifying the number of the zone to attach to.
Color - Pointer to integer value specifying the color to assign to the geometry.

0=Black	**8=Custom1**
1=Red	**9=Custom2**
2=Green	**10=Custom3**
3=Blue	**11=Custom4**
4=Cyan	**12=Custom5**
5=Yellow	**13=Custom6**
6=Purple	**14=Custom7**
7=White	**15=Custom8**

FillColor - Pointer to integer value specifying the color used to fill the geometry. See *Color* above.
IsFilled - Pointer to integer flag to specify if geometry is to be filled.

GeomType - Pointer to integer value specifying the geometry type.

0=2D Line Segments	**3=Circle**
1=Rectangle	**4=Ellipse**
2=Square	**5=3D Line Segments**

LinePattern - Pointer to integer value specifying the line pattern.

0=Solid	**3=Dotted**
1=Dashed	**4=LongDash**
2=DashDot	**5=DashDotDot**

PatternLength - Pointer to double value specifying the pattern length in frame units.
LineThickness - Pointer to double value specifying the line thickness in frame units.
NumEllipsePts - Pointer to integer value specifying the number of points to use for circles and ellipses. The value must be greater than 0.
ArrowheadStyle - Pointer to integer value specifying the arrowhead style.

0=Plain	**2=Hollow**
1=Filled	

ArrowheadAttachment - Pointer to integer value specifying where to attach arrowheads.

0=None	**2=End**
1=Beginning	**3=Both**

ArrowheadSize - Pointer to double value specifying the arrowhead size in frame units.
ArrowheadAngle - Pointer to double value specifying the arrowhead angle in degrees.
Scope - Pointer to integer value specifying the scope. 0=global, 1=local.
Clipping - Specifies whether to clip the geometry (that is, only plot the geometry within) to the viewport or the frame. 0=ClipToViewport,1=ClipToFrame.
NumSegments - Pointer to integer value specifying the number of polyline segments.
NumSegPts - Array of integer values specifying the number of points in each of the *NumSegments* segments.
XOrThetaGeomData - Array of floating-point values specifying the X-coordinates.
YOrRGeomData - Array of floating-point values specifying the Y-coordinates.
ZGeomData - Array of floating-point values specifying the Z-coordinate.
MFC - Macro function command. Must be null terminated.

Summary: Initializes the process of writing a binary data file. This must be called *first* before any other **TECIO** calls are made (except **TECFOREIGN110**). You may write to multiple files by calling **TECINI110** more than once. Each time **TECINI110** is called, a new file is opened. Use **TECFIL110** to switch between files.

FORTRAN Syntax:
```
      INTEGER*4 FUNCTION TECINI110(Title,
&                                 Variables,
&                                 FName,
&                                 ScratchDir,
&                                 Debug,
&                                 VIsDouble)
 CHARACTER*(*)  Title
 CHARACTER*(*)  Variables
 CHARACTER*(*)  FName
 CHARACTER*(*)  ScratchDir
 INTEGER*4  Debug
 INTEGER*4  VIsDouble
```
C Syntax:
```
#include TECIO.h

INTEGER4 TECINI110(char *Title,
            char *Variables,
            char *FName,
            char *ScratchDir,
            INTEGER4 *Debug
            INTEGER4 *VIsDouble);
```

Return Value: **0** if successful, **-1** if unsuccessful.

Parameters: *Title* - Title of the data set. *Must be null terminated.*
Variables - List of variable names. If a comma appears in the string it will be used as the separator between variable names, otherwise a space is used. *Must be null terminated.*
FName - Name of the file to create. *Must be null terminated.*
ScratchDir - Name of the directory to put the scratch file. *Must be null terminated.*
Debug - Pointer to the integer flag for debugging. Set to 0 for no debugging or 1 to debug. When set to 1, the debug messages will be sent to the standard output (stdout).

VIsDouble - Pointer to the integer flag for specifying whether field data generated in future calls to **TECDAT110** are to be written in single or double precision. Set to 0 for single precision or 1 for double.

TECLAB110

Summary: Write a set of custom labels to the data file.

FORTRAN Syntax:

> `INTEGER*4 FUNCTION TECLAB110(`*Labels*`)`
> `CHARACTER*(*)` *Labels*

C Syntax: `#include TECIO.h`

`INTEGER4 TECLAB110(char *`*Labels*`);`

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *Labels* - Character string of custom labels. Separate labels by a comma or space. For example, a set of custom labels for each day of the weeks is **Sun Mon Tue Wed Thu Fri Sat**.

TECNOD110

Summary: Writes an array of node data to the binary data file. This is the connectivity list for finite element zones.

FORTRAN Syntax:

> `INTEGER*4 FUNCTION TECNOD110(`*NData*`)`
> `INTEGER*4` *NData* `(`*T, M*`)`

C Syntax: `#include TECIO.h`

`INTEGER4 TECNOD110(INTEGER4 *`*NData*`);`

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *NData* - Array of integers. This is the connectivity list, dimensioned (T, M) (T moving fastest), where M is the number of elements in the zone and T is set according to the following list:

ELEMENT TYPE	*T*
Line Segment	2
Triangle	3
Quadrilateral	4
Tetrahedral	4
Brick	8

TECTXT110

Summary: Writes a text record to the data file.

FORTRAN Syntax:

```
      INTEGER*4 FUNCTION TECTXT110(XOrThetaPos,
&                                  YOrRPos,
&                                  ZOrUnusedPos,
&                                  PosCoordMode,
&                                  AttachToZone,
&                                  Zone,
&                                  Font,
&                                  FontHeightUnits,
&                                  FontHeight,
&                                  BoxType,
&                                  BoxMargin,
&                                  BoxLineThickness,
&                                  BoxColor,
&                                  BoxFillColor,
&                                  Angle,
&                                  Anchor,
&                                  LineSpacing,
&                                  TextColor,
&                                  Scope,
&                                  Clipping,
&                                  Text,
&                                  MFC)
      DOUBLE PRECISION XOrThetaPos
```

```
            DOUBLE PRECISION YOrRPos
            DOUBLE PRECISION ZOrUnusedPos,
            INTEGER*4 PosCoordMode
            INTEGER*4 AttachToZone
            INTEGER*4 Zone
            INTEGER*4 Font
            INTEGER*4 FontHeightUnits
            DOUBLE PRECISION FontHeight
            INTEGER*4 BoxType
            DOUBLE PRECISION BoxMargin
            DOUBLE PRECISION BoxLineThickness
            INTEGER*4 BoxColor
            INTEGER*4 BoxFillColor
            DOUBLE PRECISION Angle
            INTEGER*4 Anchor
            DOUBLE PRECISION LineSpacing
            INTEGER*4 TextColor
            INTEGER*4 Scope
            INTEGER*4 Clipping
            CHARACTER*(*) Text
            CHARACTER*(*) MFC
```

C Syntax: `#include TECIO.h`

```
INTEGER4 TECTXT110(double *XOrThetaPos,
                double *YOrRPosPos,
                double *ZOrUnusedPos,
                INTEGER4 *PosCoordMode,
                INTEGER4 *AttachToZone,
                INTEGER4 *Zone,
                INTEGER4 *Font,
                INTEGER4 *FontHeightUnits,
                double *FontHeight,
                INTEGER4 *BoxType,
                double *BoxMargin,
                double *BoxLineThickness,
                INTEGER4 *BoxColor,
                INTEGER4 *BoxFillColor,
                double *Angle,
                INTEGER4 *Anchor,
                double *LineSpacing,
                INTEGER4 *TextColor,
```

```
                        INTEGER4 *Scope,
                        INTEGER4 *Clipping,
                        char *Text,
                        char *MFC)
```

Return Value: 0 if successful, **-1** if unsuccessful.

Parameters: *XOrThetaPos* - Pointer to double value specifying the X-position or Theta-position (polar plots only) of the text.
YOrRPos - Pointer to double value specifying the Y-position or R-position (polar plots only) of the text.
ZOrUnusedPos - Pointer to double value specifying the Z-position of the text.
PosCoordMode - Pointer to integer value specifying the position coordinate system.

        0=Grid
        1=Frame
        6=Grid3D

AttachToZone - Pointer to integer flag for to signal that the text is "attached" to a zone.
Zone - Pointer to integer value specifying the zone number to attach to.
Font - Pointer to integer value specifying the font.

0=Helvetica	**6=Times Italic**
1=Helvetica Bold	**7=Times Bold**
2=Greek	**8=Times Italic Bold**
3=Math	**9=Courier**
4=User-Defined	**10=Courier Bold**
5=Times	

FontHeightUnits - Pointer to integer value specifying the font height units.

0=Grid	**2=Point**
1=Frame	

FontHeight - Pointer to double value specifying the font height.
BoxType - Pointer to integer value specifying the box type.

0=None	**2=Hollow**
1=Filled	

BoxMargin - Pointer to double value specifying the box margin (in frame units).
BoxLineThickness - Pointer to double value specifying the box line thickness (in frame units).

BoxColor - Pointer to integer value specifying the color to assign to the box.

0=Black	**8=Custom1**
1=Red	**9=Custom2**
2=Green	**10=Custom3**
3=Blue	**11=Custom4**
4=Cyan	**12=Custom5**
5=Yellow	**13=Custom6**
6=Purple	**14=Custom7**
7=White	**15=Custom8**

BoxFillColor - Pointer to integer value specifying the fill color to assign to the box. (See *BoxColor*)

Angle - Pointer to double value specifying the text angle in degrees.

Anchor - Pointer to integer value specifying where to anchor the text.

0=Left	**5=MidRight**
1=Center	**6=HeadLeft**
2=Right	**7=HeadCenter**
3=MidLeft	**8=HeadRight**
4=MidCenter	

LineSpacing - Pointer to double value specifying the text line spacing.

TextColor - Pointer to integer value specifying the color to assign to the text. (See *BoxColor*)

Scope - Pointer to integer value specifying the scope.

0=Global	**1=Local**

Clipping - Specifies whether to clip the geometry (that is, only plot the geometry within) to the viewport or the frame. 0=ClipToViewport,1=ClipToFrame.

Text - Character string representing text to display. *Must be null terminated.*

MFC - Macro function command. Must be null terminated.

TECUSR110

Summary: Writes a character string to the data file in a USERREC record. USERREC records are ignored by Tecplot, but may be used by add-ons.

FORTRAN Syntax:

```
      INTEGER*4 FUNCTION TECUSR110(S)
   CHARACTER*(*) S
```

C Syntax: `#include TECIO.h`

`INTEGER4 TECUSR110(CHAR *`*S*`);`

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *S* - The character string to write to the data file. Must be null-terminated.

TECVAUXSTR110

Summary: Writes an auxiliary data item to the data file for the specified variable. Must be called after **TECINI110** and before **TECEND110.** Auxiliary data may be used by text, macros, equations (if it is numeric) and add-ons. It may be viewed directly in the AuxData page of the Data Set Information dialog.

FORTRAN Syntax:

```
        INTEGER*4 FUNCTION TECVAUXSTR110(Var, Name,
Value)
&
    INTEGER*4 Value
    CHARACTER*(*) Name
    CHARACTER*(*) Value
```

C Syntax: `#include TECIO.h`

```
INTEGER4 TECZAUXSTR110(INTEGER4 *Var,
                        char *Name,
                        char *Value);
```

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *Var* - The varible number for which to set the auxiliary data. Variable numbers start at one.
Name - The name of the auxiliary data item. If a data item with this name already exists, its value will be overwritten. Must be a null-terminated character string.
Value - The auxiliary data value to be written to the data file. Must be a null-terminated character string.

TECZAUXSTR110

Summary: Writes an auxiliary data item for the current zone to the data file. Must be called after **TECZNE110** for the desired zone. Auxiliary data may be used by text,

macros, equations (if it is numeric) and add-ons. It may be viewed directly in the AuxData page of the Data Set Information dialog.

FORTRAN Syntax:

```
        INTEGER*4 FUNCTION TECZAUXSTR110(Name, Value)
&
    CHARACTER*(*) Name
    CHARACTER*(*) Value
```

C Syntax: `#include TECIO.h`

```
INTEGER4 TECZAUXSTR110(char *Name,
                              char *Value);
```

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *Name* - The name of the auxiliary data item. If a data item with this name already exists, its value will be overwritten. Must be a null-terminated character string.
Value - The auxiliary data value to be written to the data file. Must be a null-terminated character string.

TECZNE110

Summary: Writes header information about the next zone to be added to the data file. After **TECZNE110** is called, you must call **TECDAT110** one or more times (and then call **TECNOD110** if the data format is **FEBLOCK** or **FEPOINT**).

FORTRAN Syntax:

```
        INTEGER*4 FUNCTION TECZNE110(ZoneTitle,
&ZoneType,
&IMxOrNumPts,
&JMxOrNumElements,
&KMx,
&ICellMax,
&JCellMax,
&KCellMax,
&SolutionTime,
&StrandID,
&ParentZone,
&IsBlock,
&NumFaceConnections,
&FaceNeighborMode,
&PassiveVarList,
```

&*ValueLocation,*
&*ShareVarFromZone*
&*ShareConnectivityFromZone***)**

CHARACTER*(*)*ZoneTitle*
INTEGER*4*ZoneType*
INTEGER*4*IMxOrNumPts*
INTEGER*4*JMxOrNumElements*
INTEGER*4*KMx*
INTEGER*4*ICellMax*
INTEGER*4*JCellMax*
INTEGER*4*KCellMax*
DOUBLE PRECISION *Solution Time*
INTEGER*4*StrandID*
INTEGER*4*ParentZone*
INTEGER*4*N*
INTEGER*4*M*
INTEGER*4*IsBlock*
INTEGER*4*NumFaceConnections*
INTEGER*4*FaceNeighborMode*
INTEGER*4*PassiveVarList*
INTEGER*4*ValueLocation*
INTEGER*4*ShareVarFromZone*
INTEGER*4*ShareConnectivityFromZone*

C Syntax: **#include TECIO.h**

INTEGER4 TECZNE110(char **ZoneTitle***,**
INTEGER4**ZoneType***,**
INTEGER4**IMxOrNumPts***,**
INTEGER4**JMxOrNumElements***,**
INTEGER4**KMx***,**
INTEGER4**ICellMax***,**
INTEGER4**JCellMax***,**
INTEGER4**KCellMax***,**
DOUBLE**SolutionTime,*
INTEGER4**StrandID***,**
INTEGER4**ParentZone,*
INTEGER4**IsBlock***,**
INTEGER4**NumFaceConnections***,**
INTEGER4**FaceNeighborMode***,**

```
INTEGER4 *PassiveVarList,
INTEGER4 *ValueLocation,
INTEGER4 *ShareVarFromZone,
INTEGER4 *ShareConnectivityFromZone)
```

Return Value: `0` if successful, `-1` if unsuccessful.

Parameters: *ZoneTitle* - The title of the zone. Must be null-terminated.

ZoneType - The type of the zone: 0=ORDERED, 1=FELINESEG, 2=FETRIAN-GLE, 3=FEQUADRILATERAL, 4=FETETRAHEDRON, 5=FEBRICK

IMxOrNumPts - For ordered zones, the number of nodes in the I index direction. For finite-element zones, the number of nodes.

JMxOrNumElements - For ordered zones, the number of nodes in the J index direction. For finite-element zones, the number of elements.

KMx - For ordered zones, the number of nodes in the K index direction. Not used for finite-element zones.

ICellMax - For zones of type FEBRICK only, the number of cells logically connected in the I index direction.

JCellMax - For zones of type FEBRICK only, the number of cells logically connected in the J index direction.

KCellMax - For zones of type FEBRICK only, the number of cells logically connected in the K index direction.

SolutionTime - Scalar double precision value specifying the time associated with the zone.

StrandID - Scalar integer value specifying the strand to which the zone is associated. A value of zero indicates the zone is static and not associated with a strand. A value of -1 indicates that the strand id for this zone is pending assignment by Tecplot's data loader (intermediate value only). Values greater than 0 indicate a zone is assigned to a given strand.

ParentZone - Scalar integer value representing the relationship between this zone and its parent. A value of zero indicates that this zone is not associated with a parent zone. A value greater than zero is considered this zone's parent. A zone may not specify itself as its own parent. With a parent zone association, Tecplot can generate surface-restricted streamtraces.

IsBlock - Indicates whether the data will be passed into TECDAT110 in BLOCK or POINT format. 0=POINT, 1=BLOCK.

NumFaceConnections - The number of face connections that will be passed in routine TECFACE110.

FaceNeighborMode - The type of face connections that will be passed in routine TECFACE110. 0=LocalOneToOne, 1=LocalOneToMany, 2=GlobalOneToOne, 3=GlobalOneToMany

PassiveVarList - Array, dimensioned by the number of variables, of 4 byte integer values specifying the active/passive nature of each variable. A value of 0 indi-

cates the associated variable is active while a value of 1 indicates that it is pas-
sive.

ValueLocation - The location of each variable in the data set. ValueLocation(I)
indicates the location of variable I for this zone. 0=cell-centered, 1=node-cen-
tered. Pass null to indicate that all variables are node-centered.

ShareVarFromZone - Indicates variable sharing. Array, dimensioned by the num-
ber of variables. ShareVarFromZone(I) indicates the zone number with which
variable I will be shared. This reduces the amount of data to be passed via
TECDAT110. A value of 0 indicates that the variable is not shared. Pass null to
indicate no variable sharing for this zone. You must pass null for the first zone in
a data set (there is no data available to share).

ShareConnectivityFromZone - For finite-element zones only, Indicates the zone
number with which connectivity is shared. Pass 0 to indicate no connectivity
sharing. You must pass 0 for the first zone in a data set.

The commands below are the old TECIO commands which still work for purposes of backwards
compatibility. Note that in many cases, these functions take the same inputs as their current coun-
terparts.

3- 7.1 Example Programs

Example programs written in both FORTRAN and C that demonstrate the use of the TECIO utility
functions can be found in the util/tecio directory below the Tecplot Home Directory:

simtest.f, **simtest.c**:Demonstrates simple use of the TECIO utility functions.

comtest.f, **comtest.c**:Demonstrates the complex use of TECIO utility functions such as
multiple file generation and transient data.

See the file readme.txt in the same directory for instructions on how to compile these examples.