Data Format Guide
Tecplot 360 2025 Release 2

Tecplot, Inc.

Table of Contents

Introduction

Tecplot 360 can read data produced in many different formats using the data loaders provided with the product. This manual describes a complementary approach: writing your data in the Tecplot 360 data format so it can be read natively, providing the best experience for Tecplot 360 users. This Data Format Guide includes the following topics:

- Data Structure Learn about the different types of data structure available in Tecplot 360 and how to use them.
- Binary Data Refer to this chapter for details on outputting data in Tecplot 360's binary file format (`.plt`) or the newer subzone load format (`.szplt`) using the "classic" APIs in the TecIO library, a collection of routines we provide that you can use to write files in these format from your software.
- ASCII Data We strongly recommend that you create binary data files. However, we provide the ASCII data format, as it is very convenient for producing small, simple files.
- Reading SZL Data Files and Writing SZL Data Files (New API) describe using newer TecIO APIs to read and write files in Tecplot's SZL (`.szplt`) binary format.
- Binary Data File Format, documents the `.plt` binary file format.
- Glossary Refer to the Glossary for the definitions of terms used throughout the manual.

Before continuing to either the Binary or ASCII chapter, please review this overview of Best Practices.

Subzone Loading

Tecplot 360 introduced a new subzone file format with extension `.szplt` that is optimized for loading partial zones ("subzones") as individual chunks of data are needed for a plot or for other operations. Subzone loading improves performance substantially for many common operations on large cases while also reducing RAM usage. If your software generates large data files, we strongly encourage you to support this format, as it will significantly improve your user experience. You can use the same API as is used to write `.plt` files, so if your program already uses TecIO, it is straightforward to upgrade it to write `.szplt` files instead of, or in addition to, `.plt` files. For new development, we suggest using the newer API described in Writing SZL Data Files (New API).

Creating Data Files for Tecplot 360 and Focus

For the purposes of this discussion, "polyhedral" refers to either polyhedral or polygonal zones.

If you intend to create data files that will load in both Tecplot 360 and Tecplot Focus, be aware that polyhedral and polygonal zones are not supported by Tecplot Focus. If any of the zones in a given data file are polyhedral, you will not be able to load the data file into Tecplot Focus. To create data files that

will load in both products, you must use either ordered zones or "classic" (cell-based) finite element zones (triangular, quadrilateral, tetrahedral or brick elements).

The TecIO library file included with each release of Tecplot Focus is identical to the library file included with the corresponding release of Tecplot 360 and can create polyhedral zones, even though you will not be able to load the data files into Tecplot Focus. Similarly, this guide is largely the same for both products; we have included reminders throughout about this Tecplot Focus limitation.

The subzone load file format (`.szplt`) is not supported by Tecplot Focus, nor by legacy versions of Tecplot 360 (versions without "EX" in their designation). Current versions of Tecplot Focus (2016 R1 and later), derived from Tecplot 360, also limit the number of data elements in a single data set to 5 million.

Best Practices

Users who wish to generate native Tecplot 360 data files automatically from applications such as complex flow solvers have a number of options for outputting data into Tecplot's data format. This section outlines a few "best practices" for outputting your data into Tecplot 360 data format.

Create Binary Data Files (.plt or .szplt) instead of ASCII (.dat)

Binary data files are more efficient than ASCII files, in terms of disk space and time to first image. To create binary data files, you may use functions provided in the TecIO library. To create ASCII files, you can write out plain text in the usual manner.

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. If this is the case, please contact us at support@tecplot.com. There may be a way to resolve this issue.
- You wish users to be able to view or edit the data in a text editor.

Offer the Option to Write Subzone Load Files (.szplt)

- Users of Tecplot 360 will appreciate the improved experience provided by these files, especially for large, 3-dimensional, non-Polyhedral cases.
- Use `TECFLUSH142` to reduce memory usage when writing SZL files. `TECFLUSH142` must be called after writing a complete zone.
- Use SZL to allow Read capability.

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 360. If your data files are small and you can only obtain the data in a point-like format (for example, with a spreadsheet), then using point format is acceptable.

Binary files can only be written in block format. Point format is allowed for ASCII files, but running the preplot utility will convert the data to block format.

Add Auxiliary data to Preset Variable Assignments in Tecplot 360

Zone Auxiliary data can be used to give Tecplot 360 hints about properties of your data. For example, it can be used to set the defaults for which variables to use for certain kinds of plots. Auxiliary data is supported by both binary and ASCII formats. Refer to `TECAUXSTR142` or Data Set Auxiliary Data Record for information on working with auxiliary data in binary or ASCII data files, respectively.

Transient Data

- 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 disk space, as well as memory when the data is loaded into Tecplot 360. In addition, the benefits are compounded with scratch data derived from these variables because it is also shared within Tecplot 360. See also `TECZNE142` and `TECPOLYZNE142` (for binary data) or Variable and Connectivity List Sharing (for ASCII data).

- Do not share variables between 3D and 2D zones
- Consider using Grid and Solution files for transient data.
- Use one object or mesh per StrandID which consists of multiple solution times. See `TECZNE142` or `TECPOLYZNE142` for more information.

Parallel Output

- Use TecIO-MPI which writes SZL files in parallel.
- Be sure to write to a parallel file system.
- Maximize striping for efficiency.

Passive Variables

Tecplot 360 can manage many data sets at the same time. However, within a given data set 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 or not calculated. You can set selected variables in those zones to be passive. A passive variable is one that will always return the value zero if queried (e.g. in a probe) but will not involve itself in operations such as the calculations of the min and max range. This is very useful when calculating default contour levels.

Use the Native Byte Ordering for the Target Machine

When you create binary data, you can elect to produce these files in either Motorola (big-endian) byte order or Intel (little-endian) byte order. Today's most popular platforms all use Intel byte order, and

generally this is the order you should use when writing binary data. The exceptions involve older platforms no longer supported by Tecplot. If you are using such legacy platforms, it can improve performance to write the binary data in the order native to the platform on which it will be viewed, if this is known, even though Tecplot 360 will load data with either byte order. For See the notes about this option in the User's Manual for the Preplot flag.

- SZL doesn't support big-endian byte ordering and only supports Native Byte Ordering at this time. `TECFOREIGN142` has not yet been implemented in the SZL API.

Auxiliary Data Influencing Tecplot 360's Behavior

Behavior in Tecplot is influenced by the presence of particular auxiliary data assignments:

- If axis variables X, Y or Z are not assigned, the dataset auxiliary data is searched for "Common.XVar", "Common.YVar", and "Common.ZVar". If found, and one or more of the auxiliary data items contains a valid dataset variable number, it is assigned to the corresponding axis variable.
- If vector variables, U, V or W are not assigned, the dataset auxiliary data is searched for "Common.UVar", "Common.VVar", and "Common.WVar". If found, and one or more of the auxiliary data items contains a valid dataset variable number, it is assigned to the corresponding vector variable.
- If the contour variable, C, is not assigned, the dataset auxiliary data is searched for "Common.CVar". If found, and the auxiliary data item contains a valid dataset variable number, it is assigned to the contour variable.
- Fieldmaps comprised of one or more zones having an auxiliary data item named, "Common.IsBoundaryZone", with a value of "Yes", "Y", "True", "T", or "ON", are deactivated by default, unless the zone's auxiliary data also contains an item named, "Common.BoundaryCondition", with a value of "Wall".
- If a transient dataset auxiliary data item of the name "Common.TransientZoneVisibility" is present, it must be located in a dataset and its value is either "ZonesAtOrBeforeSolutionTime" or "ZonesAtSolutionTime". "Common.TransientZoneVisibilitty" determines what is shown when no zones with a given Strand ID exist at the current solution time, within a small time tolerance.
  - ZonesAtOrBeforeSolutionTime: For each strand at a given solution time, the zones shown are those that are at the solution time, within a tolerance. Except for the very last time step, if no zones exist at the given solution time, zones shown are those from the first prior solution time showing zones.
  - ZonesAtSolutionTime: For each strand at a given solution time, the zones shown are those that are at the solution time, within a tolerance.

CFDA Auxiliary Data

A number of aux data values will set values in dialogs available under the Analyze menu.

- A dataset aux data variable "Common.Incompressible" set to "Yes", "Y", "True", "T", or "ON" will set the Incompressible toggle in the Fluid Properties dialog.
- For incompressible flow, the density of the flow is taken from the value of data set aux data variable "Common.Density"
- Several aux data variables may be used to set other values in the Analyze>Fluid Properties dialog. For each of these, a constant value and a data set variable number may be specified. If only a constant value is specified, then that value is used for calculations at all nodes or cells in the data set. If a data set variable number is specified, then the value of that variable is used at each node or cell, and the constant value is used only for reference (free-stream) value calculations.

Name	Type	Description
Common.SpecificHeat	double	For incompressible flow only, the specific heat of the fluid.
Common.SpecificHeatVar	int	For incompressible flow only, the data set variable that holds the value of specific heat.
Common.GasConstant	double	For compressible flow only, the gas constant of the fluid.
Common.GasConstantVar	int	For compressible flow only, the data set variable that holds the value of the gas constant.
Common.Gamma	double	For compressible flow only, the ratio of specific heats.
Common.GammaVar	int	For compressible flow only, the data set variable that holds the value of the ratio of specific heats.
Common.Viscosity	double	The value of the dynamic viscosity of the fluid.
Common.ViscosityVar	int	The data set variable that holds the value of the fluid dynamic viscosity
Common.Conductivity	double	The value of the fluid thermal conductivity.
Common.ConductivityVar	int	The data set variable that holds the value of the fluid thermal conductivity.

- Any zones marked with a "Common.IsBoundaryZone" value of "Yes", "Y", "True", "T", or "ON" are also listed as boundaries in the Geometry and Boundaries dialog. The type of each boundary zone is taken from the zone's value of "Common.BoundaryCondition". Recognized boundary types are "Interzone", "Inflow", "Outflow", "Wall", "Slip Wall", "Symmetry", and "Extrapolated".
- In addition to the zone aux data mentioned above for boundary assignments, some dataset aux data affects settings in the Analyze>Geometry and Boundaries dialog:

Name	Type	Available Values	Description
Common.Axisymmetric	boolean	"Yes", "Y", "True", "T", or "ON"	Indicates that the solution geometry is axisymmetric.

Name	Type	Available Values	Description
Common.AxisOfSymmetryVarAssignment	option	"X" or "Y"	For axisymmetric flow, the axis variable used to define the symmetry.
Common.AxisValue	double	any	For axisymmetric flow, the value of the axis variable about which the flow is symmetric.

- The following data set aux data variables affect settings in the Analyze>Field Variables dialog. Note that only two of Common.PressureVar, Common.TemperatureVar, Common.DensityVar, Common.StagnationEnergyVar and Common.MachNumberVar should be set. Also, Common.DensityVar and Common.MachNumberVar should not be used for incompressible flow:

Name	Type	Description
Common.VectorVarsAreVelocity	boolean	Set to "Y" to Indicate that the vector vars indicated by Common.UVar, Common.VVar and Common.WVar are velocity. Otherwise, they will be assumed to be or momentum.
Common.PressureVar	double	The data set variable that holds values of pressure.
Common.TemperatureVar	double	The data set variable that holds values of temperature.
Common.DensityVar	double	For compressible flow only, the data set variable that holds values of density.
Common.StagnationEnergyVar	double	The data set variable that holds values of stagnation energy.
Common.MachNumberVar	double	For compressible flow

- Reference (free-stream) conditions may be set by the following dataset aux data variables, and will be reflected in the Analyze/Reference Values dialog. Note that Common.ReferenceMachNumber, Common.ReferenceDensity and Common.ReferenceSpeedOfSound should not be used for incompressible flow:

Name	Type	Description
Common.ReferenceU	double	The reference X component of velocity.
Common.ReferenceV	double	The reference Y component of velocity.
Common.ReferenceMachNumber	double	For compressible flow only, the reference Mach number.
Common.AngleOfAttack	double	The reference angle of attack.
Common.ReferencePressure	double	The reference pressure.

Name	Type	Description
Common.ReferenceDensity	double	For compressible flow only, the reference density.
Common.ReferenceTemperature	double	The reference temperature.
Common.ReferenceSpeedOfSound	double	For compressible flow only, the reference speed of sound.

- The following dataset aux data values may be set to identify dataset variables for turbulent flow calculations. These settings will be reflected in the Analyze>Calculate Turbulence Functions dialog. You should specify only two of these:

Name	Type	Description
Common.TurbulentKineticEnergyVar	int	The dataset variable that holds turbulent kinetic energy, commonly referred to as "k".
Common.TurbulentDissipationRateVar	int	The dataset variable that holds turbulent dissipation rate, commonly referred to as "ε".
Common.TurbulentFrequencyVar	int	The dataset variable that holds turbulent frequency, commonly referred to as "ω".
Common.TurbulentViscosityVar	int	The dataset variable that holds turbulent kinematic viscosity.
Common.TurbulentDynamicViscosityVar	int	The dataset variable that holds turbulent dynamic viscosity.

Data Structure

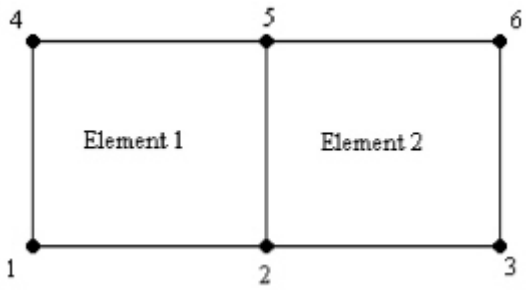
Tecplot 360 accommodates two different types of data: Ordered Data and Finite Element Data. Ordered data is a set of points logically stored in a one-, two-, or three-dimensional array, where I, J, and K are the index values within the array. The number of data points is the product of all of the dimensions within the array. Finite-element data is arranged in two arrays, a variable array and a connectivity matrix. The variable array is a collection of points in 2D or 3D space that are connected into polygonal or polyhedral units called *elements*. The connections between the nodes are defined by the connectivity matrix.

A connectivity list is used to define which nodes are included in each element of an ordered or cell-based finite element zone. You should know your zone type and the number of elements in each zone in order to create your connectivity list.

The number of nodes required for each element is implied by your zone type. For example, if you have a finite element quadrilateral zone, you will have four nodes defined for each element. Likewise, you must provide eight numbers for each cell in a BRICK zone, and three numbers for each element in a TRIANGLE zone. If you have a cell that has a smaller number of nodes than that required by your zone

type, simply repeat a node number. For example, if you are working with a finite element quadrilateral zone and you would like to create a triangular element, simply repeat a node in the list (e.g., 1,4,5,5).

In the example below, the zone contains two quadrilateral elements. Therefore, the connectivity list must have eight values. The first four values define the nodes that form Element 1. The second four values define the nodes that form Element 2.



The connectivity list for this example would appear as follows:

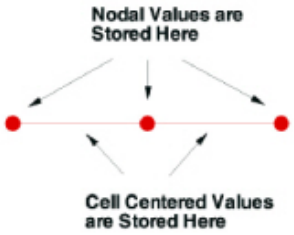
```
ConnList[8] = {4,5,2,1,       /* nodes for Element 1 */
               5,6,3,2};      /* nodes for Element 2 */
```

It is important to provide your node list in either a clockwise or counter-clockwise order. Otherwise, your cell will twist, and the element produced will be misshapen.

Ordered Data

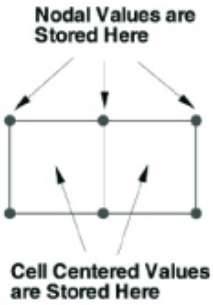
Ordered data is defined by one, two, or three-dimensional logical arrays, dimensioned by IMAX, JMAX, and KMAX. These arrays define the interconnections between nodes and cells. The variables can be either nodal or cell-centered. Nodal variables are stored at the nodes; cell-centered values are stored within the cells.

One-dimensional Ordered Data (I-ordered, J-ordered, or K-ordered)



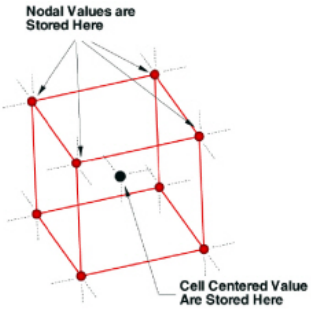
A single dimensional array where either IMAX, JMAX or KMAX is greater than or equal to one, and the others are equal to one. For nodal data, the number of stored values is equal to IMAX * JMAX * KMAX. For cell-centered I-ordered data (where IMAX is greater than one, and JMAX and KMAX are equal to one), the number of stored values is (IMAX-1) - similarly for J-ordered and K-ordered data.

Two-dimensional Ordered Data (IJ-ordered, JK-ordered, IK-ordered)



A two-dimensional array where two of the three dimensions (IMAX, JMAX, KMAX) are greater than one, and the other dimension is equal to one. For nodal data, the number of stored values is equal to IMAX * JMAX * KMAX. For cell-centered IJ-ordered data (where IMAX and JMAX are greater than one, and KMAX is equal to one), the number of stored values is (IMAX-1)(JMAX-1) - similarly for JK-ordered and IK-ordered data.

Three-dimensional Ordered Data (IJK-ordered)



A three-dimensional array where all IMAX, JMAX and KMAX are each greater than one. For nodal ordered data, the number of nodes is the product of the I-, J-, and K-dimensions. For nodal data, the number of stored values is equal to IMAX * JMAX * KMAX. For cell-centered data, the number of stored values is (IMAX-1)(JMAX-1)(KMAX-1).

Finite Element Data

While finite element data is usually associated with numerical analysis for modeling complex problems in 3D structures (heat transfer, fluid dynamics, and electromagnetics), it also provides an effective approach for organizing data points in or around complex geometrical shapes. For example, you may not have the same number of data points on different lines, there may be holes in the middle of the dataset, or the data points may be irregularly (randomly) positioned. For such difficult cases, you may be able to organize your data as a patchwork of elements. Each element can be independent of the other elements, so you can group your elements to fit complex boundaries and leave voids within sets of elements. The figure below shows how finite element data can be used to model a complex boundary.

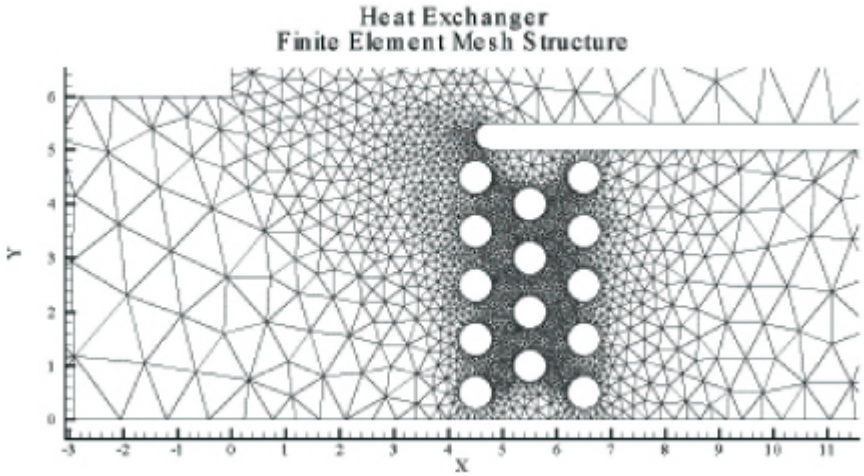


Figure 1. This figure shows finite element data used to model a complex boundary.

Finite element data defines a set of points (nodes) and the connected elements of these points. The variables may be defined either at the nodes or at the cell (element) center. Finite element data can be divided into three types:

Line data

This is a set of line segments defining a 2D or 3D line. Unlike I-ordered data, a single finite element line zone may consist of multiple disconnected sections. The values of the variables at each data point (node) are entered in the data file similarly to I-ordered data, where the nodes are numbered with the I-index. This data is followed by another set of data defining connections between nodes. This second section is often referred to as the connectivity list. All elements are lines consisting of two nodes, specified in the connectivity list.

Surface data

This is a set of triangular, quadrilateral, or polygonal elements defining a 2D field or a 3D surface. When using polygonal elements, the number of sides may vary from element to element. In finite element surface data, you can choose (by zone) to arrange your data in three point (triangle), four point (quadrilateral), or variable-point (polygonal) elements. The number of points per node and their arrangement are determined by the element type of the zone. If a mixture of quadrilaterals and triangles is necessary, you may repeat a node in the quadrilateral element type to create a triangle, or you may use polygonal elements.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Volume data

This is a set of tetrahedral, brick, or polyhedral elements defining a 3D volume field. When using polyhedral elements, the number of sides may vary from element to element. Finite element volume cells may contain four points (tetrahedron), eight points (brick), or variable points (polyhedral).

For cell-based element types (triangular, quadrilateral, tetrahedral, or brick), you can simulate zones with mixed element types by repeating nodes as necessary. For example, a triangle element can be included in a quadrilateral zone by repeating one node in the element's connectivity list, and tetrahedral, pyramidal, and prismatic elements can be included in a brick zone by repeating nodes appropriately.

Finite Element Data provides detailed information about how to format your FE data in Tecplot's data file format.

Line Data

Unlike I-ordered data, a single finite element line zone may consist of multiple disconnected sections. The values of the variables at each data point (node) are entered in the data file similarly to I-ordered data, where the nodes are numbered with the I-index. This data is followed by another set of data

defining connections between nodes. This second section is often referred to as the connectivity list. All elements are lines consisting of two nodes, specified in the connectivity list.

Surface Data

In finite element surface data, you can choose (by zone) to arrange your data in three point (triangle), four point (quadrilateral), or variable-point (polygonal) elements. The number of points per node and their arrangement are determined by the element type of the zone. If a mixture of quadrilaterals and triangles is necessary, you may repeat a node in the quadrilateral element type to create a triangle or you may use polygonal elements.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Volume Data

Finite element volume cells may contain four points (tetrahedron), eight points (brick) or a variable number of points (polyhedral). The figures below shows the arrangement of the nodes for tetrahedral and brick elements. The connectivity arrangement for polyhedral data is governed by the method in which the polyhedral facemap data is supplied.

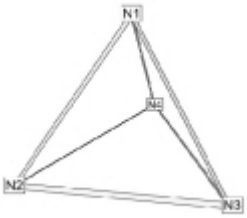


Figure 2. Tetrahedral connectivity arrangement

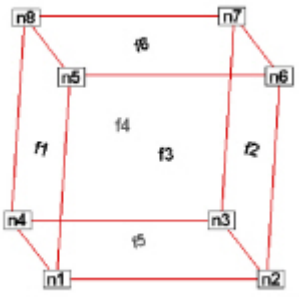


Figure 3. Brick connectivity arrangement

In the brick format, points may be repeated to achieve 4, 5, 6, or 7 point elements. For example, a connectivity list of "n1 n1 n1 n1 n5 n6 n7 n8" (where n1 is repeated four times) results in a quadrilateral-based pyramid element.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Finite Element Data Limitations

Working with finite element data has some limitations:

- XY-plots of finite element data treat the data as I-ordered; that is, the connectivity list is ignored. Only nodes are plotted, not elements, and the nodes are plotted in the order in which they appear in the data file.
- Index skipping in vector and scatter plots treats finite element data as I-ordered; the connectivity list is ignored. Nodes are skipped according to their order in the data file.

Variable Location

Data values can be stored at the nodes or at the cell centers.

- For finite element meshes, cell-centers are the centers (centroids) of elements.
- For many types of plots, cell-centered values are interpolated to the nodes internally.

Face Neighbors

For polygonal and polyhedral data, face neighbors are the fundamental means of defining the topology.

A cell is considered a neighbor if one of its faces shares all nodes in common with the selected cell, or if it is identified as a neighbor by face neighbor data in the dataset. The face numbers for cells in the various zone types are defined below.

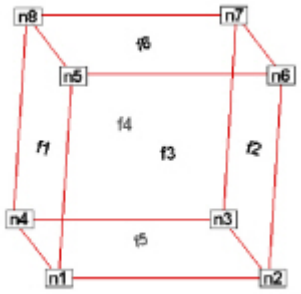


Figure 4. Example of node and face neighbors for an FE-brick cell or IJK-ordered cell.

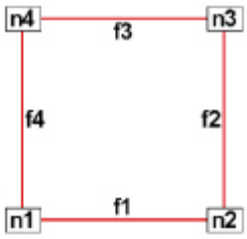


Figure 5. Example of node and face numbering for an IJ-ordered/ FE-quadrilateral cell.

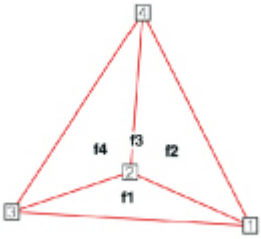


Figure 6. Example of tetrahedron face neighbors.

The implicit connections between elements in a zone may be overridden, or connections between cells in adjacent zones established by specifying face neighbor criteria in the data file. Refer to `TECFACE142` for additional information.

Working with Unorganized Data Sets

Tecplot 360 loads unorganized data as a single I-ordered zone and displays them in XY Mode, by default. Tecplot products consider an I-ordered zone irregular if it has more than one dependent variable. An I-ordered data set with one dependent variable (i.e. an XY or polar line) is NOT an irregular zone.

To check for irregular data, you can go to the **Data Set Info** dialog (accessed via the **Data** menu). The values assigned to: IMax, JMax, and KMax are displayed in the lower left quadrant of that dialog. If IMax is greater than 1, and JMax and KMax are equal to 1, then your data is irregular.

It is also possible to tell if you have irregular data by looking at the plot. If you are looking at irregular data with the Mesh layer turned on, the data points will be connected by lines in the order the points appear in the data set.

You can organize your data set for Tecplot 360 in one of the following ways:

1. Manually order the data file using a text editor.
2. Use one of the **Data→Interpolation** options.

Example - Unorganized Three-Dimensional Volume

To use 3D volume irregular data in field plots, you must interpolate the data onto a regular, IJK-ordered zone. To interpolate your data, perform the following steps:

1. Place your 3D volume irregular data into an I-ordered zone in a data file.
2. Read in your data file and create a 3D scatter plot.
3. From the **Data** menu, choose **Create Zone**>**Rectangular**. (**Circular** will also work.)
4. In the **Create Rectangular Zone** dialog, enter the I-, J-, and K-dimensions for the new zone; at a minimum, you should enter 10 for each dimension. The higher the dimensions, the finer the interpolation grid, but the longer the interpolating and plotting time.
5. Enter the minimum and maximum X, Y, and Z values for the new zone. The default values are the minimums and maximums of the current (irregular) dataset.
6. Click [Create] to create the new zone, and [Close] to dismiss the dialog.
7. From the **Data** menu, choose **Interpolate**>**Inverse Distance**. (**Linear** or **Kriging** also works.)
8. In the **Inverse-Distance Interpolation** dialog, choose the irregular data zone as the source zone, and the newly created IJK-ordered zone as the destination zone. Set any other parameters as desired.
9. Select the **[Compute]** button to perform the interpolation.

Once the interpolation is complete, you can plot the new IJK-ordered zone as any other 3D volume zone. You may plot iso-surfaces, volume streamtraces, and so forth. At this point, you may want to deactivate or delete the original irregular zone so as not to conflict with plots of the new zone.

Figure 7 shows an example of irregular data interpolated into an IJK-ordered zone, with iso-surfaces plotted on the resultant zone.

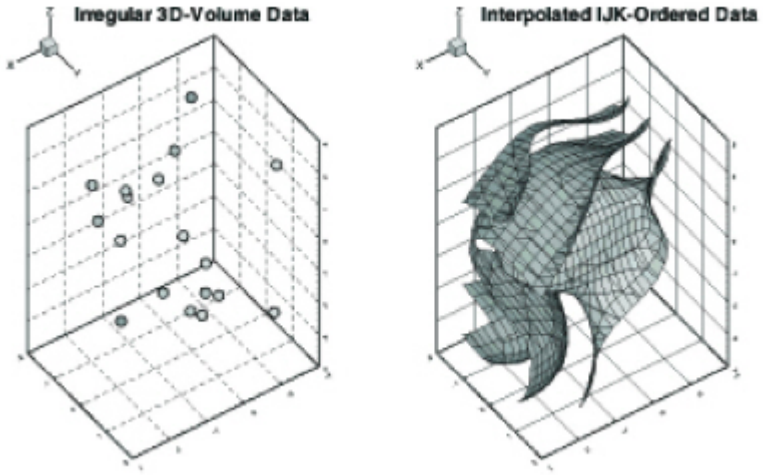


Figure 7. Irregular data interpolated into an IJK-ordered zone.

Time and Date Representation

Tecplot 360 uses floating point numbers to represent times and dates. The integer portion represents the number of days since December 30, 1899. The decimal portion represents a fractional portion of a day. The table below illustrates some examples of this method.

Date	Time	Floating Point Number
1900-01-01	00:00:00	2.0
1900-01-01	12:00:00	2.5
2008-07-31	00:00:00	39660.0
2008-07-31	12:00:00	39660.5
2008-07-31	12:01:00	39660.5006944444
2008-07-31	13:00:00	39660.5416666667

Tecplot 360 supports dates from 1800-01-01 through 9999-12-31. This formatting matches the representation method used by Microsoft Excel, enabling you to load time/date data easily from Excel into Tecplot 360. However, because Excel software's original formatting incorrectly calculated 1900 as a leap year, only dates from Mar 1, 1900 forward will import correctly into Tecplot 360.

Binary Data

This chapter is intended for experienced programmers who need to create Tecplot binary data files directly. Support for topics discussed in this chapter is limited to general questions about writing Tecplot binary files. It is beyond the scope of our Technical Support to offer programming advice or to debug programs.

It is easy to write ASCII files in text format, and they have the advantage that you can inspect them using a text editor to make sure they are being written correctly. Their primary disadvantages are that they can consume much more disk space than binary files and are slower to load, which is especially noticeable when they are large. While users can convert them to the binary format with the Preplot utility (see Converting ASCII to Binary for additional information), it is much more efficient to simply write them in binary format to begin with.

To output your data directly into Tecplot's basic binary file format, `.plt`, you may use the TecIO library, which is provided at no cost by Tecplot, Inc., or you may write your own binary functions. If you wish to write your own functions, refer to Binary Data File Format for details on the structure of `.plt` files. If you wish to link with the library provided by Tecplot, begin with Getting Started and use Binary Data File Format only for reference.

If you wish to write files in the newer `.szplt` format, you must use the TecIO library, as this file format is currently not documented.

You can find source files for most of the examples in this chapter in the `util/tecio/examples` folder of your Tecplot 360 installation.

New TecIO Writing API

This chapter describes the "classic" TecIO binary file API. If you will be writing SZL `(.szplt)` files, you may want to use the new API described in Writing SZL Data Files (New API). However, even if you use the new API, you should still read this chapter for background if you are not already familiar with the classic API.

Getting Started

TecIO is a library of utility functions that your application can use to create binary data files directly, bypassing the use of ASCII files. This makes for fewer files to manage, conserves disk space, and saves the time required to convert the files to binary (whether as a separate step or when loading them).

Tecplot products support two binary file formats:

Tecplot Binary (`.plt`)

The legacy format written by versions of Tecplot 360 and Tecplot Focus prior to Tecplot 360 EX. It is of course also supported by Tecplot 360 EX and can be read by all versions of Tecplot 360, 360 EX, and Focus.

Tecplot Subzone (`.szplt`)

A newer format introduced with Tecplot 360 EX, optimized for large data sets, that enables substantially improved interactive performance and a reduced memory footprint for common workflows. It can only be read by Tecplot 360 EX. Subzone data does not support Polyhedral data.

We encourage you to support both formats. Users of Tecplot 360 EX will appreciate the improved interactive experience, while users of Tecplot Focus, legacy versions of Tecplot 360, and other programs that read only Tecplot-format binary files will appreciate being able to consume your data with the software they use every day.

Two versions of the TecIO library are available: a standard (single-process) library and a multiprocess library that employs MPI (Message Passing Interface, a standardized, portable message-passing system available on a variety of parallel computing architectures). The MPI version only writes subzone `(.szplt)` files. Copies of both are installed with your Tecplot 360 installation. The path of the file varies depending on the platform.

Platform	Standard TecIO	Multiprocess TecIO
Linux	lib/libtecio.so	lib/libteciompi.so
Mac	tecio/libtecio.dylib	tecio/libteciompi.dylib
Windows	lib/tecio.dll	lib/libteciompi.dll

You may also download the latest version of TecIO from the Tecplot Web site. (This may be a newer version than the one described in this document.) Tecplot 360 and Focus both contain precompiled versions of TecIO in their base installation. The download from the website only contains the source files.

Before preparing to output your data in Tecplot's binary format using the `TecIO` library, we recommend you proceed as follows:

1. Review Data Structure for information on how zones and data are structured, Binary Data File Function Calling Sequence and Writing to Multiple Binary Data Files.
2. Review the example files in the `examples/tecio` folder. The example programs demonstrate the use of the **TecIO** utility functions and are provided in both FORTRAN and C/C++:

   - `simtest.f`, `simtest.f90`, `simtest.cpp` - These files demonstrate simple use of the TecIO utility functions.
   - `comtest.f`, `comtest.f90`, `comtest.cpp` - These files demonstrate complex use of TecIO utility functions, such as multiple file generation and transient data.

   Numerous additional, more modern examples included in the TecIO package target specific actions, like writing polyhedral data. Review these examples for additional guidance.
3. Follow the instructions in Linking with the TecIO Library for information on setting up your project to develop with TecIO and linking with the library.
4. Begin developing your code.

Be sure to check out the examples located in your installation directory. A brief overview of the types of examples are listed below.

Example	Description
Complex Example	comtest, create multiple plt files containing multiple zones (structured/unstructured/XY Line), also creates geometries and text.
FE (Unstructured) Face Neighbors	faceneighbors, creates plt file of simple squares with face neighbor information.
FE (Unstructured) Partitioned	brickpartitioned, creates an Unstructured .szplt file with partitioned zones, uses TecIO MPI, can be passed flags to output the partitioned zone in grid and solution .szplt files.
Ordered	ij_ordered, creates two, small, 2D, ordered zones with contour variable information.
Ordered Partitioned	ijkpartitioned, creates a .szplt file with an IJK ordered zone in 4 partitions, uses TecIO MPI.
Read/Write Szplt	rewriteszl, creates a utility that reads in a .szplt file and writes it out to another .szplt file

Example	Description
Text	text, creates a text box in a .plt file.

Table 1. TecIO Examples and Descriptions

Viewing Your Output

You may load your binary files in Tecplot 360 or Tecplot Focus using the Tecplot Data loader or the Tecplot Subzone Data Loader, as appropriate. Once loaded, you may view information about your data file using any of the following techniques:

Data Set Information dialog

You may use the **Data Set Information** dialog (accessed via the **Data** menu) to display information about your file (once it is loaded into Tecplot). Refer to this dialog for a list of the zones, variables, variable ranges, auxiliary data and more. Refer to the User's Manual for details.

Data Spreadsheet

Use the Data Spreadsheet to view a table of every variable value in your file. Refer to the User's Manual for details.

Partitioned Data

The TecIO library supports partitioning finite-element and ordered zones when writing `.szplt` data files. That is, the data does not need to be written all at once, but can be written in sections, and by more than one process. This capability is intended primarily for use with CFD codes that compute large solutions in parallel, with multiple solver processes running on independent nodes in a compute cluster, each solving a subset of the simulation. However, this capability can also be used by a single-process solver running on a desktop workstation if desired.

TecIO writes the partitioned zones into a single Tecplot `.szplt` file that can later be opened and visualized by Tecplot 360. To enable this functionality on large multiprocessing systems, TecIO utilizes the Message-Passing Interface (MPI) library.

By comparison to non-partitioned zones, only a couple of additional TecIO function calls are required when writing partitioned zones, and only a couple more beyond that when writing in parallel using MPI . Two versions of the TecIO library are provided: `tecio` and `teciompi`. The latter requires a compatible MPI library. (Note that non-MPI `tecio` does support writing partitioned zones, just not in parallel.)

The `teciompi` library only writes SZL `(.szplt)` files; it does not write traditional Tecplot binary `(.plt)` files.

Partitioned solution of large CFD cases typically requires that zone partitions overlap slightly. That is, the nodes or cells on the each side of the boundary between partitions will reside in more than one solver process: in the one that "owns" it according to the partitioning rules, as well as in any processes

solving a partition spatially adjacent to it. In processes other than the one that "owns" them, such data are commonly referred to as "ghost cells" and "ghost nodes."

The TecIO library needs information about these overlapping cells and nodes in order to later allow the partitioned data to be joined into a single virtual data set suitable for visualization. For finite-element zones, each solver process must pass a list of its ghost nodes and cells along with the zone data (which should include the ghost nodes and cells). For ordered zones, the solver need pass only the nodes that form non-ghost cells, which provides sufficient information for later reassembling the data.

The code sample `brickpartitioned` is useful for understanding how to write partitioned zones, and includes the additional function calls needed for writing partitioned data in parallel using MPI. The ijkpartitioned code sample shows this for an IJK-ordered zone, and also demonstrates outputting non-partitioned zones from various MPI ranks.

Binary File Compatibility

A `.plt` file is compatible with the version of Tecplot 360 EX or Focus tied to the version of the TecIO library that you use and later versions. For example, if you use the TecIO library that was bundled with Tecplot 360 EX 2014 R2, your files can be loaded with Tecplot 360 EX 2014 R2 and newer. A `.plt` file is also backward compatible to the first version of Tecplot 360 (pre-EX) or Focus that uses the file format version supported by the library being used.

This rule is independent of the version number used for the binary functions (for example, the 142 in TECINI). Even if you use 142 functions with the version of the TecIO library included with this distribution, your `.plt` file will be compatible with this version of Tecplot 360 EX and newer.

However, older Tecplot products cannot read `.szplt` files regardless of their version. Subzone data files (SZL or `.szplt)` can be loaded only in Tecplot 360 EX, not Tecplot 360 (pre-EX) or Tecplot Focus.

The SZL file format was introduced in Tecplot 360 EX 2014 R1 and was revised in Tecplot 360 EX 2016 R1 to support partitioned output. Similar to the situation with the `.plt` format, newer versions of Tecplot 360 EX can read the format generated with the older TecIO library, but older versions of Tecplot 360 EX may not always be able to read the files generated by the newer versions of TecIO.

Files containing face-based (polygonal or polyhedral) finite-element zones cannot be read by Tecplot Focus. Tecplot Focus 2016 R1 and later can read files containing a maximum of five million data points.

However, the TecIO shipped with Tecplot EX 2014 R2 and later will produce files compatible with older versions dating back to 2009 as long as the file does not contain polyhedral or polygonal zones with more than 2 billion face nodes.

Deprecated Binary Functions

Functions whose names end in an integer less than 142 are deprecated and are provided only for compatibility with older client code. We recommend you use the 142 binary function family with new code and/or if you need to update your application to take advantage of the new functionality provided

with version 142. In order to use the 142 family of functions, use the TecIO library included in your Tecplot 360 EX 2025 distribution. If you have existing code using deprecated functions, and want to use any binary function calls from version 142, you must update all your TecIO library calls to 142.

API version 142 or later allows applications to select between the `.plt` and `.szplt` file formats at runtime, a feature introduced with Tecplot 360 2014 R2. In Tecplot 360 2014 R1, two versions of the TecIO library were provided, one that wrote `.plt` files and one that wrote `.szplt` files. Both had identical APIs; the file format was determined solely by the version of the library you linked with your application. In Tecplot 360 2014 R2 and later, a parameter was added to TECINI to choose the format when opening the file for writing (see `TECINI142`. The current TecIO library always writes `.plt` files when using an API version below 142, since there is no way to specify the file format with these older APIs.

Character Strings in FORTRAN

All character string parameters passed to TecIO must use C-style strings: that is, they must terminate with a null character. In FORTRAN, this can be 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`, use the following syntax:

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

Boolean Flags

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

Binary Data File Function Calling Sequence

The binary data file functions in the TecIO library must be called in a specific order, which varies slightly depending on the type of zone being written and whether you are using the serial version (tecio) or parallel version (teciompi) of the library. Remember, you may write both partitioned and non-partitioned zones with either version of the TecIO library; the MPI version is required only if you want to write files in parallel.

Even if your code generally supports writing partitioned zones, single-partition zones are considered non-partitioned by TecIO, and you should use the non-partitioned zone calling sequence.

A program may, if desired, write both ordinary and partitioned data to up to ten files simultaneously. Writing to Multiple Binary Data Files explains how to use the `TECFIL142` function to write to multiple files simultaneously.

Additional Requirement for Parallel Output

For parallel programs using TecIO-MPI, additional routines must be called and some restrictions must be observed:

- The routine TECMPIINIT142 must be called after each call to TECINI142. TECMPIINIT142 identifies an MPI communicator that includes all MPI ranks (processes) involved in outputting the file, and designates one rank within the communicator group as the main rank. It must be called by all members of the communicator. If more than one file is being output-TECINI142 is called more than once-each file output may use a different communicator and a different main rank within that communicator.
- Meta-data and other non-zone items may be output from only the main rank for a file. Thus, the routines TECAUXSTR142, TECVAUXSTR142, TECZAUXSTR142, TECLAB142, TECGEO142, and TECTXT142 can be called only from the main rank. TECUSR142 has no effect because TecIO-MPI outputs only .szplt files, and user data is not included in the .szplt file format.
- For each zone output, TECZNE142 or TECZNEFEMIXED142 must be called by the main rank and all ranks outputting that zone. It is optional for uninvolved ranks to also call TECZNE142 or TECZNEFEMIXED142; it becomes a no-op for these ranks.
- TECZNEMAP142 must be called immediately after the call to TECZNE142 or TECZNEFEMIXED142 (including any uninvolved ranks that still called TECZNE142 or TECZNEFEMIXED142). For partitioned zones, TECZNEMAP142 indicates how many partitions will be output for the zone, and identifies the MPI rank that will output each partition. For non-partitioned zones, TECZNEMAP142 is called as if the zone had a single partition. This gives a mechanism for your program to designate, for each non-partitioned zone, which rank will output that zone.

Non-partitioned Zones

When writing non-partitioned zones to a file, calls should be made to TecIO in this order:

- TECINI142, **then** TECMPIINIT142 if using TecIO-MPI
- For each call to TECINI142, use one or more of the following:
  - TECAUXSTR142 *(Optional, main rank only if using TecIO-MPI)*
  - TECVAUXSTR142 *Optional, main rank only if using TecIO-MPI)*
  - TECZNE142 **or** TECPOLYZNE142 **or** TECZNEFEMIXED142 **then** TECZNEMAP142 if using TecIO-MPI
  - Then, for each call to TECZNE142 **or** TECPOLYZNE142 **or** TECZNEFEMIXED142, use one or more of these:
    - TECDAT142 *(One or more to fill each zone)*
    - TECNOD142 **or** TECNODE142 *(One or more for each finite element zone)*
    - TECFACE142 *(One for each zone with face connections)*
    - TECPOLYFACE142 **and** TECPOLYBCONN142 *(polyhedral data; not available for TecIO-MPI)*
    - TECZAUXSTR142 *(Optional, main rank only if using TecIO-MPI)*

- TECLAB142 *(Optional, main rank only if using TecIO-MPI)*
- TECGEO142 *(Optional, main rank only if using TecIO-MPI)*
- TECTXT142 *(Optional, main rank only if using TecIO-MPI)*
- TECUSR142 *(Optional, main rank only if using TecIO-MPI)*

- TECEND142

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Partitioned Zones

Currently, only classic volume finite-element zones and IJK-ordered zones can be written in partitioned fashion. Other zone types are planned for future development.

The calling sequence for writing partitioned zones is as follows:

- TECINI142, then TECMPIINIT142 if using TecIO-MPI
- TECAUXSTR142 *(main rank only)*
- TECVAUXSTR142 *(main rank only)*
- Then for each partition or zone to be written by the process, use one or more of the following:
  - TECZNE142 **or** TECZNEFEMIXED142, **then** TECZNEMAP142 (if using TecIO-MPI) followed by either TECFEPTN142, TECIJKPTN142, **or** TECFEMIXEDPTN142,depending on whether the zone is FE single-section, ordered, or FE multi-section.
    - Then call one or more of the following:
    - TECDAT142 *(One or more calls to fill each zone)*
    - TECNOD142 **or** TECNODE142 *(One or more calls for each finite element zone)*
    - TECFACE142 *(One call for each zone with face connections)*
    - TECZAUXSTR142 *(main process only)*
  - TECLAB142 *(main rank only)*
  - TECGEO142 *(main rank only)*
  - TECTXT142 *(main rank only)*
  - TECUSR142 *(main rank only)*
- TECEND142

Writing to Multiple Binary Data Files

Each time TECINI142 is called, it sets up a new file context. For each file context, you must maintain the order of the calls as described earlier. The TECFIL142 function is used to switch between file contexts. Up to 10 files can be written to at a time. TECFIL142 can be called almost anywhere after TECINI142 has

been called. The only parameter to TECFIL142, an integer, *n*, shifts the file context to the *nth* open file. The files are numbered relative to the order of the calls to TECINI142.

Note that when writing a subzone (.szplt) file, all data written is held in memory and not actually committed to disk until TECEND142 is called, because the order of data stored in a subzone file is usually not the same as the order in which it was provided by the application. Ordinary binary files (.plt) files may be partially written to disk before TECEND142 is called.

Linking with the TecIO Library

Follow the instructions below to link with the TecIO library. The library is provided as a dynamic library on all platforms, meaning that it remains a separate file and must be distributed with your application. If you wish instead to use a static library that will be combined into your application, you may obtain the TecIO or TecIO-MPI source code from my.tecplot.com/portal/product-releases/tecio-library. For TecIO-MPI on Mac or Linux platforms, you should strongly consider building the library from source on your solver machine, since various MPI versions are not binary-compatible.

Linux/Macintosh

To link with the TecIO library, pass the full path to the tecio (or teciompi) library to your compiler or linker along with all other input files needed to compile and link your application. The TecIO library is written in C++, so in addition to linking it, you will likely also need to link in the C++ standard library.

For example, to create an output file my-executable from a C source file of my-prog.c and link in the TecIO library and the C++ standard library:

```
cc -o my-executable my-prog.c /path/to/libtecio.so -lstdc++ (Linux)
cc -o my-executable my-prog.c /path/to/libtecio.dylib -lstdc++  (Mac)
```

For TecIO-MPI, instead use /path/to/libteciompi.so or /path/to/libteciompi.dylib depending on platform.

The stdc++ library used to build the tecio library in your Tecplot 360 installation may be newer than the one provided on your platform. If so, link against the version of the stdc++ library provided in your Tecplot 360 installation. See the file util/tecio/examples/base.make inside your Tecplot 360 installation directory for an example of this.

Include the TecIO header file TECIO.h in your source files. It may be found in the include directory of your Tecplot 360 installation.

Fortran programmers: some Fortran 90 compilers do not recognize the .f90 filename extension.

Windows

To link with the TecIO library, list `tecio.dll` (or, for TecIO-MPI, `teciompi.dll)` as an additional dependency in your Visual Studio project. Include the TecIO header file `TECIO.h` in your source files. It may be found in the `include` directory of your Tecplot 360 installation.

To ensure Windows finds `tecio.dll` when launching your executable, ensure its location is in your `PATH` environment variable, or else copy it to your executable's directory. The latter approach is generally best, as it ensures that the correct version will be used if multiple copies of the library are installed on the machine.

Notes for Windows Programmers using Fortran

Files `tecio.f90` and `tecio.for`, located in the `include` folder in your installation, contain both Fortran-90 interfaces for all `TecIO` routines and several compiler-specific directives (the `!MS$ATTRIBUTES` lines). These direct Microsoft Visual Fortran to use `STDCALL` calling conventions with by-reference parameter passing. While `tecio.f90` is free-formatted, `tecio.for` uses the traditional column-based formatting. Include the appropriate file in any of your subroutines that call `TecIO` routines. Both files were developed for Intel Visual Fortran 9.

Users of other compilers may need to adjust the Fortran settings or add other compiler directives to achieve the same effect. In particular, Fortran strings must be null-terminated and passed without a length argument.

Binary Data File Function Reference

This section describes each of the TecIO functions in detail.

TECAUXSTR142

Writes auxiliary data for the data set to the data file. The function may be called at any time between `TECINI142` and `TECEND142`. Auxiliary data may be used by text, macros, equations (if it is numeric) and add-ons. It may be viewed directly in the **Aux Data** page of the **Data Set Information** dialog (accessed via the **Data** menu).

When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECAUXSTR142(Name,
&                                Value)
 CHARACTER*(*) Name
 CHARACTER*(*) Value
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECAUXSTR142(char *Name,
                      char *Value);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
Name	The name of the auxiliary data. If this duplicates an existing name, the value will overwrite the existing value. It must be a null-terminated character string and cannot contain spaces.
Value	The value to assign to the named auxiliary data. It must be a null-terminated character string.

Example

For example, to set an Auxiliary Variable called DeformationValue to 0.98:

```
char DeformationValue[128];
strcpy(DeformationValue,"0.98");

TECAUXSTR142("DeformationValue",
             DeformationValue);
```

When the data file is loaded into Tecplot, "Deformation Value" will appear on the **Aux Page** of the **Data Set Information** dialog when "for Data Set" is selected in **Show Auxiliary Data** menu.

TECDAT142

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 TECZNE142 or TECPOLYZNE142 or TECZNEFEMIXED142).

TECDAT142 allows you to write your data in piecemeal fashion in case it is not contained in one contiguous block in your program or is not available all at once. TECDAT142 must be called enough times to ensure that the correct number of values is written for each zone and that the aggregate order for the data is correct.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECDAT142(N,
&                             Data,
&                             IsDouble)
 INTEGER*4                    N
 REAL or DOUBLE PRECISION     Data(1)
 INTEGER*4                    IsDouble
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECDAT142(INTEGER4  *N,
                   void  *Data,
                   INTEGER4  *IsDouble);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
N	Pointer to an integer value specifying number of values to write.
Data	Array of single or double precision data values. Refer to Table 2 for a description of how to arrange your data.
IsDouble	Pointer to the integer flag stating whether the array Data is single (0) or double (1) precision.

Data Arrangement

The following table describes the order the data must be supplied given different zone types. VarLocation is a parameter supplied to `TECZNE142` or `TECPOLYZNE142` or `TECZNEFEMIXED142`.

Zone Type	Var. Location	Number of Values	Order

Ordered	Nodal	IMax*
		JMax*
		KMax*
		NumVars
			I varies fastest, then J, then K, then Vars. That is, the numbers should be supplied in the following order:
			`for (Var=1;Var<=NumVars;Var++)`
			`  for (K=1;K<=KMax;K++)`
			`  for (J=1;J<=JMax;J++)`
			`  for (I=1;I<=IMax;I++)`
			`  Data[I, J, K, Var] = value;`

Ordered	Cell Centered	(IMax-1)*
		(JMax-1)*
		(KMax-1)*
		NumVars
			I varies fastest, then J, then K, then Vars. That is, the numbers should be supplied in the following order:
			`for (Var=1;Var<=NumVars;Var++)`
			`  for (K=1;K<=(KMax-1);K++)`
			`  for (J=1;J<=(JMax-1);J++)`
			`  for (I=1;I<=(IMax-1);I++)`
			`  Data[I, J, K, Var] = value;`

Finite element	Nodal	IMax (i.e. NumNodes) * NumVars	N varies fastest, then Vars. That is, the numbers should be supplied in the following order:
			`for (Var=1;Var<=NumVars;Var++)`
			`  for (N=1;N<=NumNodes;N++)`
			`  Data[N, Var] = value;`

Finite element	Cell Centered	JMax (i.e. NumElements) * NumVars	E varies fastest, then Var. That is, the numbers should be supplied in the following order:
			`for (Var=1;Var<=NumVars;Var++)`
			`  for (E=1;E<=NumElements;E++)`
			`  Data[E, Var] = value;`

Table 2. Data Arrangement

Refer to Examples for examples using `TECDAT142`:

- Face Neighbors
- Polygonal Example

- Multiple Polyhedral Zones
- Multiple Polygonal Zones
- Polyhedral Example
- IJ-ordered zone

TECEND142

Must be called to close the current data file. There must be one call to `TECEND142` for each `TECINI142` or data may be lost. (When writing `.szplt` files, all data are held in memory until `TECEND142` is called.)

When writing partitioned data, this call will block until all processes involved in writing the data have called it.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECEND142()
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECEND142();
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

None.

TECFACE142

Writes face connections for the current zone to the file. Face Neighbor Connections are used for ordered or cell-based finite element zones to specify connections that are not explicitly defined by the connectivity list or ordered zone structure. You many use face neighbors to specify connections between zones (global connections) or connections within zones (local connections). Face neighbor connections are used by Tecplot when deriving variables or drawing contour lines. Specifying face neighbors typically leads to smoother connections. NOTE: face neighbors have expensive performance implications. Use face neighbors only to manually specify connections that are not defined via the connectivity list.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

This function must be called after TECNOD142 or TECNODE142, and may only be called if a non-zero value of *NumFaceConnections* was used in the previous call to TECZNE142 or TECZNEFEMIXED142.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECFACE142(FaceConnections)
INTEGER*4 FACECONNECTIONS(*)
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECFACE142(INTEGER4 *FaceConnections);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
FaceConnections	The array that specifies the face connections. The array must have L values, where L is the sum of the number of values for each face neighbor connection in the data file. The number of values in a face neighbor connection is dependent upon the FaceNeighborMode parameter (set via TECZNE142 or TECZNEFEMIXED142) and is described in the following table.

FaceNeighbor Mode	Number of values	Data
LocalOneToOne	3	cz1,fz,cz2
LocalOneToMany	nz+4	cz1,fz,oz,nz,cz2,cz3,...,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 360 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 360's auto face neighbor generator (FE only).

The face numbers for cells in the various zone types are shown below:

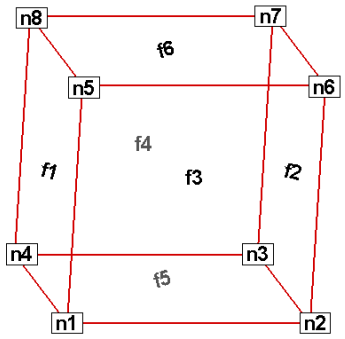


Figure 8. Example of node and face neighbors for an FE-brick cell or IJK-ordered cell.

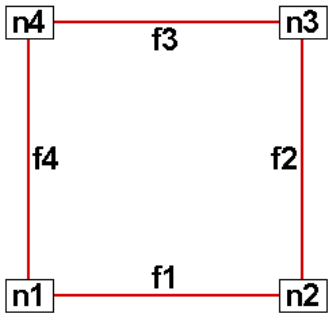


Figure 9. Example of node and face numbering for an IJ-ordered/ FE-quadrilateral cell.

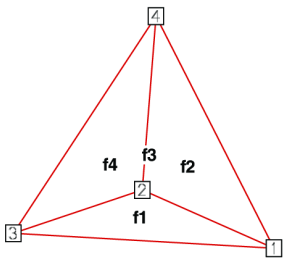


Figure 10. Example of tetrahedron face neighbors.

Example

Refer to Face Neighbors for an example of working with face neighbors. In this example, face

neighbors are used to prevent an Edge line from being drawn between the two zones.

TECFEMIXEDPTN142

When writing a partitioned FE-mixed element zone to a .szplt file, TECFEMIXEDPTN142 provides information about the partition about to be written. Must be called after TECZNEFEMIXED142 (and, for TecIO-MPI, TECZNEMAP142) but before calling TECDAT142 and TECNOD142/ TECNODE142 to actually write the data. Should not be called for non-partitioned zones (or equivalently, zones that have only one partition).

A partition may include nodes and cells that overlap other partitions, commonly referred to as "ghost nodes" and "ghost cells." The TECZNEFEMIXED142 function call specifies this ghost data, allowing the partitions to be later reassembled seamlessly into a single zone when loaded for visualization. Each node or cell should be considered to be "owned" by one process, and that process should not report that node or cell as a "ghost." Any other process that has that node or cell in its partition should include it in the list of ghost nodes or cells, respectively, passed to TECZNEFEMIXED142. The array containing the number of cells per section and the number of ghost cells per section must be dimensioned by the number of sections for the zone. All partitions of the zone must deliver both arrays dimensioned by the number of sections for the zone. If a partition does not have cells in a given section it must set the number of cells for that section to zero. The ghostCells array is a flat array containing the ghost cells for each section, back to back, interpreted by looking at the counts in the numGhostCellsPerSection array.

All partitions must include data for the actual "ghost" nodes and cells using the appropriate function calls (e.g. TECDAT142, TECNOD142/ TECNODE142).

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECFEMIXEDPTN142(partition
&                                    numNodes,
&                                    numCellsPerSection,
&                                    numGhostNodes,
&                                    ghostNodes,
&                                    neighborPartitions,
&                                    neighborPartitionNodes,
&                                    numGhostCellsPerSection,
&                                    ghostCells)
 INTEGER*4 partition
 INTEGER*8 numNodes
 INTEGER*8 numCellsPerSection
 INTEGER*8 numGhostNodes
 INTEGER*4 ghostNodes
 INTEGER*4 neighborPartitions
 INTEGER*4 neighborPartitionNodes
 INTEGER*8 numGhostCellsPerSection
```

```
 INTEGER*4 ghostCells
```

C Syntax:

```
#include "TECIO.h"
INTEGER4 TECFEMIXEDPTN142(INTEGER4 const* partition,
                          INTEGER8 const* numNodes,
                          INTEGER8 const* numCellsPerSection,
                          INTEGER8 const* numGhostNodes,
                          INTEGER4 const* ghostNodes,
                          INTEGER4 const* neighborPartitions,
                          INTEGER4 const* neighborPartitionNodes,
                          INTEGER8 const* numGhostCellsPerSection,
                          INTEGER4 const* ghostCells)
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
partition	The number of the current partition. Partitions are numbered from 1 to the number of partitions (as specified in the `TECZNEMAP142` call) inclusive.
numNodes	The number of nodes in the partition, including any ghost nodes.
numCellsPerSection	The number of cells in each section of the partition, including any ghost cells. The array must be dimensioned by the number of sections in the zone and is the same dimension for all partitions. If a partition does not have any cells in a given section a value of zero should be set for that section.
numGhostNodes	The number of ghost nodes in the partition.
ghostNodes	Array of length numGhostNodes specifying the indices of the ghost nodes. Each ghost node must be specified exactly once.
neighborPartitions	Array of length numGhostNodes specifying the owning partition of each ghost node, in the order in which they are specified in ghostNodes.
neighborPartitionNodes	Array of length `ngnodes` specifying the node index by which each ghost node is known in its owning partition, again in the order specified in ghostNodes.
numGhostCellsPerSection	The number of ghost cells in each section of the partition. The array must be dimensioned by the number of sections in the zone and is the same dimension for all partitions. If a partition does not have any ghost cells in a given section a value of zero should be set for that section.

Parameter	Description
ghostCells	Flat array of a length that is the sum of the members of the numGhostCellsPerSection array containing the ghost cells for each section of the partition listed back to back.

TECFEPTN142

When writing a partitioned classic finite-element zone to a `.szplt` file, provides information about the partition about to be written. Must be called after TECZNE142 or TECZNEMAP142(and, for TecIO-MPI, TECZNEMAP142) but before calling TECDAT142 and TECNOD142/TECNODE142 to actually write the data. Should *not* be called for non-partitioned zones (or equivalently, zones that have only one partition).

A partition may include nodes and cells that overlap other partitions, commonly referred to as "ghost nodes" and "ghost cells." The TECFEPTN142 function call specifies this ghost data, allowing the partitions to be later reassembled seamlessly into a single zone when loaded for visualization. Each node or cell should be considered to be "owned" by one process, and that process should not report that node or cell as a "ghost." Any other process that has that node or cell in its partition should include it in the list of ghost nodes or cells, respectively, passed to TECFEPTN142.

All partitions must include data for the actual "ghost" nodes and cells using the appropriate function calls (e.g. TECDAT142, TECNOD142/TECNODE142).

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECFEPTN142( partition,
&                                numnodes,
&                                numcells,
&                                ngnodes,
&                                gnodes,
&                                gnpartitions,
&                                gnpnodes,
&                                ngcells,
&                                gcells)
INTEGER*4  partition
INTEGER*4  numnodes
INTEGER*4  numcells
INTEGER*4  ngnodes
INTEGER*4  gnodes
INTEGER*4  gnpartitions
INTEGER*4  gnpnodes
INTEGER*4  ngcells
INTEGER*4  gcells
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECFEPTN142(INTEGER4 *partition,
                     INTEGER4 *numnodes,
                     INTEGER4 *numcells,
                     INTEGER4 *ngnodes,
                     INTEGER4 *gnodes,
                     INTEGER4 *gnpartitions,
                     INTEGER4 *gnpnodes,
                     INTEGER4 *ngcells,
                     INTEGER4 *gcells);
```

Return Value

0 if successful, nonzero if unsuccessful.

Parameters

Parameter	Description
partition	The number of the current partition. Partitions are numbered from 1 to the number of partitions (as specified in the TECZNEMAP142 call) inclusive.
numnodes	The number of nodes in the partition, including any ghost nodes.
numcells	The number of cells in the partition, including any ghost cells.
ngnodes	The number of ghost nodes in the partition.
gnodes	Array of length ngnodes specifying the indices of the ghost nodes. Each ghost node must be specified exactly once.
gnpartitions	Array of length ngnodes specifying the owning partition of each ghost node, in the order in which they are specified in gnodes.
gnpnodes	Array of length ngnodes specifying the node index by which each ghost node is known in its owning partition, again in the order specified in gnodes.
ngcells	Number of ghost cells.
gcells	Array of length ngcells specifying the indices of the ghost cells. Each ghost cell must be specified exactly once.

TECFIL142

Switch output context to a different file. Each time TECINI142 is called, a new file context is created. This allows you to write multiple data files concurrently. When working with multiple files, be sure to call TECFIL142 each time you wish to write to a file to ensure your data is written to the expected file.

FORTRAN Syntax:

```
`INTEGER*4 FUNCTION TECFIL142(F)
 INTEGER*4    F
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECFIL142(INTEGER4 *F);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
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 TECINI142.

Examples

Refer to Switching Between Two Files for a simple example of working with TECFIL142.

TECFLUSH142

Optional, implemented for SZL file output only. May be called at any point where the current zone data and connectivity (if any) are complete. This call will append current data to six temporary files (creating the files if they do not already exist), and release associated Tecio memory. The names of these files are the filename supplied to TECINI142, appended with each of six file suffixes: .szhdr, .szdat, .szaux, .sztxt, .szgeo and .szlab. These six files, along with any data not yet flushed to disk, will be assembled into the final .szplt file when TECEND142 is called.

If using TecIO-MPI, this is a collective function - it must be called by all processes in the communicator that was supplied in the call to TECMPIINIT142, and will block until all processes have called it.

This routine is useful for unsteady solvers that write multiple time steps of data to disk and wish to write them all to a single .szplt file. It prevents Tecio from caching all of those time steps in memory, which might exhaust all available memory on the solver machine. It is an alternative to grid/solution files, and should not be used when writing grid/solution files because it adds unnecessary processing overhead for no benefit.

At any time after the first call to TECFLUSH142, the temporary files may be assembled manually using shell utility szcombine. Please see the documentation for SZCOMBINE for more information.

The data can be left in the temporary files, and even appended to in subsequent solver runs, by not calling TECEND142 at the end of the solver run. To append to existing temporary files in a new solver run, pass the same file name to TECINI142 as was used to create the temporary files in the first solver run. Note that the time steps of the new solver run will not be able to share variables or connectivity with any time steps from previous solver runs.

FORTRAN Syntax:

```
 INTEGER*4 TECFLUSH142(NumZonesToRetain,
&                      ZonesToRetain)
 INTEGER*4 NumZonesToRetain
 INTEGER*4 ZonesToRetain(*)
```

C Syntax:

```
INTEGER4 TECFLUSH142(INTEGER4 const* NumZonesToRetain,
                     INTEGER4 const* ZonesToRetain);
```

Return value:

0 if successful, -1 otherwise.

Parameters

Parameters	Description
NumZonesToRetain	The number of zones to retain in Tecio memory for sharing from subsequent zones.
ZonesToRetain	The list of zones to retain in Tecio memory. These zones will be written to the temporary files, but will also be retained in memory so that they are available for variable or connectivity sharing by zones to be output later. These are the only zone numbers that may be referenced by the parameters ShareVarFromZone or ShareConnectivityFromZone in subsequent calls to TECZNE142 or TECZNEFEMIXED142.

TECFOREIGN142

Optional function that sets the byte ordering request for subsequent calls to TECINI142. The byte ordering request will remain in effect until the next call to this function. This has no effect on any files already opened via TECINI142. Use this function to reverse the byte ordering from the format native to your operating system. This function is not much needed today, since current Tecplot products are supported only on Intel-based platforms; however, it may be useful with older versions of TecIO running on legacy UNIX platforms that have non-Intel byte orders.

If the function call is omitted, native byte ordering is used.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECFOREIGN142(DoForeignByteOrder)
 INTEGER*4                        DoForeignByteOrder
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECFOREIGN142(INTEGER4 *DoForeignByteOrder);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
DoForeignByte Order	Pointer to boolean value indicating if future files created by TECINI142 should be written out in foreign byte order. 0 indicates native byte order. 1 indicates foreign byte order.#

TECGEO142

Adds a geometry object to the file (e.g. a circle or a square). You cannot set unused parameters to NULL; use dummy values for unused parameters.

When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECGEO142(XOrThetaPos,
&                             YOrRPos,
&                             ZPos,
&                             PosCoordMode,
&                             AttachToZone,
&                             Zone,
&                             Color,
&                             FillColor,
&                             IsFilled,
&                             GeomType,
&                             LinePattern,
&                             PatternLength,
&                             LineThicknessness,
&                             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            LineThicknessness
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 TECGEO142(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    *LineThicknessness,
                    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

Parameter	Description
XPos or ThetaPos	Pointer to double value specifying the X- position or, for polar line plots, the Theta-position of the geometry.
YPos or RPos	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 Grid3D is available only when the GeomType is equal to 3D Line Segments.
AttachToZone	Pointer to integer flag to signal that the geometry is "attached" to a zone. When a geometry is attached to a zone, it will be visible only when that zone is visible. 1 = Yes 0 = No

Parameter	Description
Zone	Pointer to integer value specifying the number of the zone to attach to. Must be greater than or equal to one.
Color	Pointer to integer value specifying the color to assign to the geometry. 0=Black 1=Red 2=Green 3=Blue 4=Cyan 5=Yellow 6=Purple 7=White 8=Custom1 9=Custom2 10=Custom3 11=Custom4 12=Custom5 13=Custom6 14=Custom7 15=Custom8
FillColor	Pointer to integer value specifying the color used to fill the geometry. Refer to Color for a list of available values.
IsFilled	Pointer to integer flag to specify if geometry is to be filled. 1 = Yes 0 = No
GeomType	Pointer to integer value specifying the geometry type. 0=2D Line Segments 1=Rectangle 2=Square 3=Circle 4=Ellipse 5=3D Line Segments
LinePattern	Pointer to integer value specifying the line pattern. 0=Solid 1=Dashed 2=DashDot 3=Dotted 4=LongDash 5=DashDotDot
PatternLength	Pointer to double value specifying the pattern length in frame units (from 0.01 and less than 100).
LineThicknessness	Pointer to double value specifying the line thickness in frame units. The value must be greater than 0.0001 and less than 100.
NumEllipsePts	Pointer to integer value specifying the number of points to use for circles and ellipses. The value must be between 2 and 720.
ArrowheadStyle	Pointer to integer value specifying the arrowhead style. 0=Plain 2=Hollow 1=Filled

Parameter	Description
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 (from 0 to 100).
ArrowheadAngle	Pointer to double value specifying the arrowhead angle in degrees.
Scope	Pointer to integer value specifying the scope with respect to frames. A local scope places the object in the active frame. A global scope places the object in all frames that contain the active frame's data set. 0=Global 1=Local.
Clipping	Specifies whether to clip the geometry (that is, only plot the geometry within 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.
XGeomData	Array of floating-point values specifying the X-, Y- and Z-coordinates. Refer to Data Values for information regarding the values required for each GeomType.
ThetaGeomData	
YGeomData	
RGeomData	
ZGeomData	
MFC	Macro function command. Must be null terminated.

Origin positions

The origin (**XOrThetaPos**, **YOrRPos**, **ZPos**) of each geometry type is listed below:

- `SQUARE`- lower left corner at **XOrThetaPos**, **YOrRPos**.
- `RECTANGLE`- lower left corner at **XOrThetaPos**, **YOrRPos**.
- `CIRCLE` - centered at **XOrThetaPos**, **YOrRPos**.
- `ELLIPSE`- centered at **XOrThetaPos**, **YOrRPos**.
- `LINE` - anchored at **XOrThetaPos**, **YOrRPos**.
- `LINE3D`- anchored at **XOrThetaPos**, **YOrRPos**, **ZPos**.

Data Values

The origin (**XOrThetaGeomData**, **YOrRGeomData**, **ZGeomData**) of each geometry type is listed below:

- `SQUARE`- set **XOrThetaGeomData** equal to the desired length.
- `RECTANGLE`- set **XOrThetaGeomData** equal to the desired width and **YOrThetaGeomData** equal to the desired height.
- `CIRCLE` - set **XOrThetaGeomData** equal to the desired radius.
- `ELLIPSE`- set **XOrThetaGeomData** equal to the desired width along the x-axis and **YOrThetaGeomData** equal to the desired width along the y-axis.
- `LINE` - specify the coordinate positions for the data points in each line segment with *XOrThetaGeomData * and **YOrRGeomData**.
- `LINE3D`- specify the coordinate positions for the data points in each line segment with **XOrThetaGeomData**, **YOrRGeomData** and **ZGeomData**.

TECIJKPTN142

When writing a partitioned IJK-ordered zone to a `.szplt` file, `TECIJKPTN142` provides information about the partition about to be written. Must be called after `TECZNE142` (and, for TecIO-MPI, `TECZNEMAP142`) but before calling `TECDAT142` to actually write the data. Should not be called for non-partitioned zones (or equivalently, zones that have only one partition).

The `TECIJKPTN142` function specifies the I, J, and K indices of the partition (excluding any "ghost" cells), allowing the partitions to be later reassembled seamlessly into a single zone when loaded for visualization. The indices passed refer to indices of the entire IJK-ordered zone. Therefore, the I dimension (for example) of a partition is `imax - imin + 1`.

Index ranges of neighboring partitions must exactly coincide: for example, if one zone's `imax` is 25, the `imin` of the partition immediately to the right must be 25.

Data subsequently written using `TECDAT142` must output only the data corresponding to the nodes and cells indicated by this range.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECIJKPTN142(partition,
&                                imin,
&                                jmin,
&                                kmin,
&                                imax,
&                                jmax,
&                                kmax)
 INTEGER*4                       partition
```

```
 INTEGER*4                      imin
 INTEGER*4                      jmin
 INTEGER*4                      kmin
 INTEGER*4                      imax
 INTEGER*4                      jmax
 INTEGER*4                      kmax
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECIJKPTN(INTEGER4 *partition,
                   INTEGER4 *imin,
                   INTEGER4 *jmin,
                   INTEGER4 *kmin,
                   INTEGER4 *imax,
                   INTEGER4 *jmax,
                   INTEGER4 *kmax);
```

Return Value

0 if successful, nonzero if unsuccessful.

Parameters

Parameter	Description
partition	The number of the current partition. Partitions are numbered from 1 to the number of partitions (as specified in the TECZNEMAP142 call inclusive).
imin	The smallest I index in the partition.
jmin	The smallest J index in the partition.
kmin	The smallest K index in the partition.
imax	The largest I index in the partition.
jmax	The largest J index in the partition.
kmax	The largest K index in the partition.

TECINI142

Initializes the process of writing a binary data file. This function must be called *first* before any other TecIO calls are made (except TECFOREIGN142).

You may write to multiple files by calling TECINI142 more than once. Each time TECINI142 is called, a new file is created and a new context established for it. Use TECFIL142 to switch between files. For each call to TECINI142, there must be a corresponding call to TECEND142.

If using TecIO-MPI, the `TECINI142` call must be followed immediately by a call to `TECMPIINIT142`.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECINI142(Title,
&                             Variables,
&                             FName,
&                             ScratchDir,
&                             FileFormat,
&                             FileType,
&                             Debug,
&                             VIsDouble)
 CHARACTER*(*)                Title
 CHARACTER*(*)                Variables
 CHARACTER*(*)                ScratchDir
 CHARACTER*(*)                FName
 INTEGER*4                    FileFormat
 INTEGER*4                    FileType
 INTEGER*4                    Debug
 INTEGER*4                    VIsDouble
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECINI142(char  *Title,
                   char  *Variables,
                   char  *FName,
                   char *ScratchDir,
                   INTEGER4 *FileFormat,
                   INTEGER4 *FileType,
                   INTEGER4 *Debug
                   INTEGER4 *VIsDouble);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
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.*

Parameter	Description
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.*
FileFormat	Specifies the file format to be used. Ignored by TecIO-MPI, which always writes `.szplt` files. 0=Tecplot binary (.plt) 1=Tecplot subzone (.szplt)
FileType	Specify whether the file is a full data file (containing both grid and solution data), a grid file or a solution file. 0=Full 1=Grid 2=Solution
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 `TECDAT142` are to be written in single or double precision. 0=Single 1=Double

Examples

Each example in Examples calls `TECINI142` at least once. Refer to this section for details.

TECLAB142

Adds custom labels to the data file. Custom Labels can be used for axis labels, legend text, and tick mark labels. 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. You must have at least one zone in your data set.

A custom label set is added to your file each time you call `TECLAB142`. You may have up to sixty labels in a set and up to ten sets in a file. Each label must be surrounded by double-quotes, e.g. "Mon" "Tues" "Wed", etc. The `\n` escape sequence may be used to indicate a line break.

Custom labels are assigned to an object via the Tecplot interface. Refer to User's Manual for details.

When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECLAB142(Labels)
CHARACTER*(*)               Labels
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECLAB142(char *Labels);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
Labels	Character string of custom labels. Each label must be surrounded by double-quotes. Separate labels by a comma or space. You may have up to sixty labels in each call to TECLAB142.

Examples

To add the days of the week to your data file, to be displayed along the x-axis:

```
char Labels[60] = "\"Mon\", \"Tues\",\"Wed\",\"Thurs\", \"Fri\"";
TECLAB142(&Labels[0]);
```

TECMPIINIT142

Initializes MPI and joins a specified MPI communicator. This is a collective function. Must be called immediately after TECINI142 by all processes in the supplied communicator, and will block until all processes have called it. If processes call TECINI142 multiple times to create multiple files, TECMPIINIT142 must be called immediately after each call to TECINI142. All processes may then switch output among the open files by calling TECFIL142 as usual.

The mainrank process is the only process that may output non-zone data, such as text, geometries and aux data.

For TecIO-MPI only; does not exist in standard TecIO library.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECMPIINIT142(communicator, mainrank)
INTEGER*4 communicator
INTEGER*4 mainrank
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECMPIINIT142(void* communicator,
                       INTEGER4 const* mainrank);
```

Return Value

0 if successful, nonzero if error.

Parameters

Parameter	Description
communicator	Pointer to MPI communicator. May be MPI_COMM_WORLD or any other MPI_Comm that includes all MPI processes involved in output.
mainrank	The ID of the process (rank) within the communicator designated as the main process. Must be the same for all calling processes.

TECNOD142

Writes an array of node data to the binary data file. This is the connectivity list for cell-based finite element zones (line segment, triangle, quadrilateral, brick, and tetrahedral zones). The connectivity list for face-based finite element zones (polygonal and polyhedral) is specified via TECPOLYFACE142.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Note that node data are not stored in solution files, so do not call TECNOD142 if the file type specified in TECINI142 was SOLUTION.

See also TECNODE142, which allows you to provide connectivity information in arbitrarily-sized chunks rather than requiring it all at once.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECNOD142(NData)
INTEGER*4 NData (T, M)
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECNOD142(INTEGER4 *NData);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
NData	Array of integers listing the nodes for each element. 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: 2=Line Segment 3=Triangle 4=Quadrilateral 4=Tetrahedral 8=Brick

Examples:

Refer to Face Neighbors for examples using TECNOD142.

TECNODE142

Writes a chunk of node data to the binary data file. This is the connectivity list for cell-based finite element zones (line segment, triangle, quadrilateral, brick, and tetrahedral zones). The connectivity list for face-based finite element zones (polygonal and polyhedral) is specified via TECPOLYFACE142.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Note that node data are not stored in solution files, so do not call TECNODE142 if the file type specified in TECINI142 was SOLUTION.

This function is similar to TECNOD142 but does not require that the entire connectivity list be provided at once. Rather, you may call TECNODE142 as many times as you like, providing connectivity information for as many elements as you like each time, so long as you eventually provide connectivity information for all elements in the zone.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECNODE142(N, NData)
INTEGER*4 N
INTEGER*4 NData (T, M)
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECNODE142(INTEGER4 *N,
                    INTEGER4 *NData);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
N	Pointer to an integer indicating the number of values to write.
NData	Array of integers listing the nodes for each element. This is the connectivity list, dimensioned (T, N) (*T* moving fastest), where N is the number of elements provided in this call to TECNODE142 and T is set according to the following list: 2=Line Segment 3=Triangle 4=Quadrilateral 4=Tetrahedral 8=Brick

TECPOLYFACE142

Writes the face nodes of the face map for polygonal and polyhedral zones. All numbering schemes are one-based. The first node is Node 1, the first Face is Face 1, and so forth. Refer to Defining Polyhedral and Polygonal Data for additional information.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

This function may be called any number of times, with any number of face nodes each time, so long as face nodes for all faces are eventually written. You must also, at some point, call TECPOLYBCONN142 to specify any boundary connections in the zone; this can be done in any order, even to the point of interleaving calls to specify boundary connections and face nodes.

Note that face data are not stored in solution files, so do not call TECPOLYFACE142 if the file type specified in TECINI142 was SOLUTION.

Avoid creating concave objects (or bad meshes), as they will not look good when plotted.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECPOLYFACE142(
&                                  NumFaces,
&                                  FaceNodeCounts,
&                                  FaceNodes,
&                                  FaceLeftElems,
&                                  FaceRightElems)
 INTEGER*4                         NumFaces(*)
 INTEGER*4                         FaceNodeCounts(*)
 INTEGER*4                         FaceNodes(*)
 INTEGER*4                         FaceLeftElems(*)
 INTEGER*4                         FaceRightElems(*)
```

C Syntax:

```
#include TECIO.h
INTEGER4
TECPOLYFACE142(INTEGER4 *NumFaces,
               INTEGER4 *FaceNodeCounts,
               INTEGER4 *FaceNodes,
               INTEGER4 *FaceLeftElems,
               INTEGER4 *FaceRightElems);
```

Return Value

0 if successful; -1 if unsuccessful.

Parameters

Parameter	Description
NumFaces	The number of faces being defined in this call. TECPOLYFACE142 may be called any number of times with any number of faces in each call, so long as all faces in the zone are eventually defined.
FaceNodeCounts	An array used to define the number of nodes in each face. The array is dimensioned by NumFaces. This is NULL for polygonal zones, as each face in a polygonal zone is already known to have exactly two nodes.
FaceNodes	An array used to specify the nodes belonging to each face. The array is dimensioned by the sum of the FaceNodeCounts array for polyhedral zones or, for polygonal zones, twice NumFaces.
FaceLeftElems	An array used to define the left neighboring element for each face. The array is dimensioned by NumFaces.

Parameter	Description
FaceRightElems	An array used to define the right neighboring element for each face. The array is dimensioned by NumFaces.

Examples

Refer to the following sections for examples using TECPOLYFACE142:

- Polygonal Example
- Multiple Polyhedral Zones
- Multiple Polygonal Zones
- Polyhedral Example

TECPOLYBCONN142

Writes the boundary connections of the face map for polygonal and polyhedral zones. Boundary faces are faces that either have more than one neighboring cell on a side or have at least one neighboring cell in another zone. (Refer to Boundary Faces and Boundary Connections for a simple example.)

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

All numbering schemes are one-based. The first node is Node 1, the first face is Face 1, and so forth. Refer to Defining Polyhedral and Polygonal Data for additional information.

This function may be called any number of times, with any number of boundary connections each time, so long as boundary connections for all faces are eventually written. You must also, at some point, call TECPOLYFACE142 at least once to specify the face nodes. This can be done in any order, even to the point of interleaving calls to specify boundary connections and face nodes.

Note that connection data are not stored in solution files, so do not call TECPOLYBCONN142 if the file type specified in TECINI142 was SOLUTION.

Avoid creating concave objects (or bad meshes), as they will not look good when plotted.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECPOLYBCONN142(
&                                   NumBndryFaces,
&                                   FaceBndryConnectionCounts,
&                                   FaceBndryConnectionElems,
&                                   FaceBndryConnectionZones)
 INTEGER*4                          NumBndryFaces(*)
 INTEGER*4                          FaceBndryConnectionCounts(*)
 INTEGER*4                          FaceBndryConnectionElems(*)
```

```
INTEGER*2                       FaceBndryConnectionZones(*)
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECPOLYBCONN142(INTEGER4 *NumBndryFaces,
                         INTEGER4 *FaceBndryConnectionCounts,
                         INTEGER4 *FaceBndryConnectionElems,
                         INTEGER4 *FaceBndryConnectionZones);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
NumBndryFaces	The number of boundary faces being defined in this call. Each call to TECPOLYBCONN142 may define any number of boundary faces, so long as all boundary faces (i.e., NumConnectedBoundaryFaces in TECZNE142 or TECPOLYZNE142) are eventually defined.
FaceBndryConnectionCounts	An array used to define the number of boundary connections for each boundary face. The array is dimensioned by NumBndryFaces.
FaceBndryConnectionElems	An array used to define the boundary element(s) to which each boundary face is connected.
FaceBndryConnectionZones	An array used to define the zone(s) to which each boundary element belongs.

Examples

Refer to the following sections for examples using TECPOLYBCONN142:

- Polygonal Example
- Multiple Polyhedral Zones
- Multiple Polygonal Zones
- Polyhedral Example

TECPOLYZNE142

This function is provided as an alternative to calling TECZNE142 for face-based finite-element zones (FEPOLYGON or FEPOLYHEDRON). You are encouraged to use this function instead of TECZNE142or such zone. This function contains only parameters relevant to face-based zones and also supports zones

with number of faces or number of face nodes exceeding 32-bit limitations (beyond about two billion). TECPOLYZNE142 writes header information about the next face-based zone to be added to the data file. This function is not currently available in TecIO-MPI, because face-based zones are not currently supported by the .szplt file format. After TECPOLYZNE142 is called, you must call TECDAT142 one or more times, then call TECPOLYFACE142 one or more times, and call TECPOLYBCONN142 one or more times if NumConnectedBoundaryFaces is non-zero..

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECPOLYZNE142(ZoneTitle,
&                                 ZoneType,
&                                 NumNodes,
&                                 NumCells,
&                                 NumFaces,
&                                 TotalNumFaceNodes,
&                                 SolutionTime,
&                                 StrandID,
&                                 ParentZone,
&                                 NumConnectedBoundaryFaces,
&                                 TotalNumBoundaryConnections,
&                                 PassiveVarList,
&                                 ValueLocation,
&                                 ShareVarFromZone,
&                                 ShareConnectivityFromZone)
 CHARACTER*(*)                    ZoneTitle
 INTEGER*4                        ZoneType
 INTEGER*4                        NumNodes
 INTEGER*4                        NumCells
 INTEGER*8                        NumFaces
 INTEGER*8                        TotalNumFaceNodes
 DOUBLE PRECISION                 SolutionTime
 INTEGER*4                        StrandID
 INTEGER*4                        ParentZone
 INTEGER*4                        NumConnectedBoundaryFaces
 INTEGER*4                        TotalNumBoundaryConnections
 INTEGER*4                        PassiveVarList(*)
 INTEGER*4                        ValueLocation(*)
 INTEGER*4                        ShareVarFromZone(*)
 INTEGER*4                        ShareConnectivityFromZone
```

C Syntax:

```
#include TECIO.h
```

```
INTEGER4 TECPOLYZNE142(char  *ZoneTitle,
                       INTEGER4 *ZoneType,
                       INTEGER4 *NumNodes,
                       INTEGER4 *NumCells,
                       INTEGER8 *NumFaces,
                       INTEGER8 *TotalNumFaceNodes,
                       double  *SolutionTime,
                       INTEGER4 *StrandID,
                       INTEGER4 *ParentZone,
                       INTEGER4 *NumConnectedBoundaryFaces,
                       INTEGER4 *TotalNumBoundaryConnections,
                       INTEGER4 *PassiveVarList,
                       INTEGER4 *ValueLocation,
                       INTEGER4 *ShareVarFromZone,
                       INTEGER4 *ShareConnectivityFromZone)
```

Return Value

0 if successful; -1 if unsuccessful.

Parameters

Parameter	Notes
ZoneTitle	The title of the zone. Must be null-terminated.
ZoneType	The type of the zone: 6=FEPOLYGON 7=FEPOLYHEDRON
NumNodes	The number of nodes.
NumCells	The number of elements.
NumFaces	The number of faces.
SolutionTime	Scalar double precision value specifying the time associated with the zone. Refer to User's Manual for additional information on working with transient data.
StrandID	Scalar integer value specifying the strand to which the zone is associated. 0 = static zone, not associated with a strand. Values greater than 0 indicate a zone is assigned to a given strand. Refer to User's Manual for additional information on strands. If you are converting your function calls from function calls 109 or older, use "0" for StrandID.
ParentZone	ParentZone is no longer used. Enter 0 for this value.

Parameter	Notes
TotalNumFaceNodes	Total number of nodes for all faces. It is also the sum of the FaceNodeCounts array (defined in TECPOLYFACE142). For polygonal zones this value is equivalent to 2 * NumFaces. NumFaces = the number of faces in the zone. Refer to FaceNodeCounts and FaceNodes for simple example.
NumConnectedBoundaryFaces	Total number of boundary faces, where boundary faces are faces that either have more than one neighboring cell on a side or have a neighboring cell from another zone. Refer to Boundary Faces and Boundary Connections for simple example.
TotalNumBoundaryConnections	Total number of boundary connections for all faces. In general, TotalNumBoundaryConnections will be equal to NumConnectedBoundaryFaces. However, TotalNumBoundaryConnections must be greater than or equal to NumConnectedBoundaryFaces. Refer to Boundary Faces and Boundary Connections for simple example.
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 indicates the associated variable is active while a value of 1 indicates that it is passive. If all variables are active, you may pass NULL rather than an array of zeroes. Refer to [introduction/best-practices/passive-variables] for additional information.
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-centered. Pass null to indicate that all variables are node-centered.
ShareVarFromZone	Indicates variable sharing. Array, dimensioned by the number 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 TECDAT142. 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	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. NOTE: Connectivity cannot be shared between cell-based and face-based finite element zones.

Examples

Refer to the following sections for examples using TECPOLYZNE142:

- Polygonal Example
- Multiple Polyhedral Zones
- Multiple Polygonal Zones
- Polyhedral Example

TECTXT142

Adds a text box to the file. When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECTXT142(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 TECTXT142(double  *XOrThetaPos,
                   double  *YOrRPos,
                   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

Parameter	Description
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.

Parameter	Description
PosCoordMode	Pointer to integer value specifying the position coordinate system. 0=Grid 1=Frame 6=Grid3D If you use Grid3D, the plot type must be set to 3D Cartesian to view your text box.
AttachToZone	Pointer to integer flag 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 1=Helvetica Bold 2=Greek 3=Math 4=User-Defined 5=Times 6=Times Italic 7=Times Bold 8=Times Italic Bold 9=Courier 10=Courier Bold
FontHeightUnits	Pointer to integer value specifying the font height units. 0=Grid 1=Frame 2=Point
FontHeight	Pointer to double value specifying the font height. If PosCoordMode is set to FRAME, the value range is zero to 100.
BoxType	Pointer to integer value specifying the box type. 0=None 1=Filled 2=Hollow
BoxMargin	Pointer to double value specifying the box margin (in frame units ranging from 0 to 100).
BoxLineThickness	Pointer to double value specifying the box line thickness (in frame units ranging from 0.0001 to 100).
BoxColor	Pointer to integer value specifying the color to assign to the box. 0=Black 1=Red 2=Green 3=Blue 4=Cyan 5=Yellow 6=Purple 7=White 8=Custom1 9=Custom2 10=Custom3 11=Custom4 12=Custom5 13=Custom6 14=Custom7 15=Custom8

Parameter	Description
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 1=Center 2=Right 3=MidLeft 4=MidCenter 5=MidRight 6=HeadLeft 7=HeadCenter 8=HeadRight
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 with respect to frames. A local scope places the object in the active frame. A global scope places the object in all frames that contain the active frame's data set. 0=Global 1=Local
Clipping	Specifies whether to clip the text (that is, only plot the text within 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.

Examples

Refer to Text Example for an example of working with `TECTXT142`.

TECUSR142

Writes a character string to the data file in a USERREC record. USERREC records are ignored by Tecplot 360, but may be used by add-ons.

When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECUSR142(S)
CHARACTER*(*) S
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECUSR142(CHAR *S);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
S	The character string to write to the data file. Must be null-terminated.

TECVAUXSTR142

Writes an auxiliary data item to the data file for the specified variable. Must be called after TECINI142 and before TECEND142. Auxiliary data may be used by text, macros, equations (if the data is numeric) and add-ons. It may be viewed directly in the **Aux Data** page of the **Data Set Information** dialog (accessed via the **Data** menu). The value can be verified by selecting "Variable" from the "Show Auxiliary Data" menu and selecting the corresponding variable number from the menu.

When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECVAUXSTR142(Var, Name, Value)
 INTEGER*4          Var
 CHARACTER*(*)      Name
 CHARACTER*(*)      Value
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECAUXSTR142(INTEGER4 *Var,
                      char *Name,
                      char *Value);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
Var	The variable 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 and cannot contain spaces.
Value	The auxiliary data value to be written to the data file. Must be a null-terminated character string.

Example:

The following example illustrates adding auxiliary data to the pressure variable in the data file. In this case, pressure is the third variable.

```
INTEGER4 Var                    = 3;
char     PressureUnitsName[16]  = "PressureUnits";
char     PressureUnitsValue[16] = "Pascal (Pa)";

TECVAUXSTR142(&Var,
              &PressureUnitsName[0],
              &PressureUnitsValue[0]);
```

TECZAUXSTR142

Writes an auxiliary data item for the current zone to the data file. Must be called immediately after TECZNE142 or TECPOLYZNE142 or TECZNEFEMIXED142for 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 **Aux Data** page of the **Data Set Information** dialog (accessed via the **Data** menu). The value can be verified by selecting "Zone" from the "Show Auxiliary Data" menu and selecting the corresponding zone number.

When using TecIO-MPI, may only be called from the main process.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECZAUXSTR142(Name, Value)
 CHARACTER*(*)      Name
 CHARACTER*(*)      Value
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECZAUXSTR142(char *Name,
```

```
                        char *Value);
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
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 and cannot contain spaces.
Value	The auxiliary data value to be written to the data file. Must be a null-terminated character string.

Example:

The following example code adds auxiliary data to the zone. NOTE: TECZAUXSTR142 must be called immediately after TECZNE142 or TECPOLYZNE142 or TECZNEFEMIXED142for the desired zone.

```
char  CreatedByName[16]   = "CreatedBy";
char  CreatedByValue[16]  = "My Company";

TECZAUXSTR142(&CreatedByName[0],
              &CreatedByValue[0]);
```

TECZNE142

Writes header information about the next zone to be added to the data file. For face-based finite-element zones, you are encouraged to use TECPOLYZNE142 instead because that function is specifically designed for those zones and has features that TECZNE142 does not support (specifically support for more than two-billion faces and/or face nodes). For writing mixed-element or higher-order-element zones, you must instead call TECZNEFEMIXED142. When using TecIO-MPI, TECZNE142 should be immediately followed by a call to TECZNEMAP142.

After TECZNE142 (and TECZNEMAP142, if necessary) is called, you must call TECDAT142 one or more times. If the zone is a finite element zone, call TECNOD142/ TECNODE142 (cell-based zones) or TECPOLYFACE142 /TECPOLYBCONN142 (face-based zones) after calling TECDAT142.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECZNE142(ZoneTitle,
&                             ZoneType,
&                             IMxOrNumPts,
&                             JMxOrNumElements,
&                             KMxOrNumFaces,
&                             ICellMax,
&                             JCellMax,
&                             KCellMax,
&                             SolutionTime,
&                             StrandID,
&                             ParentZone,
&                             IsBlock,
&                             NumFaceConnections,
&                             FaceNeighborMode,
&                             TotalNumFaceNodes,
&                             NumConnectedBoundaryFaces,
&                             TotalNumBoundaryConnections,
&                             PassiveVarList,
&                             ValueLocation,
&                             ShareVarFromZone,
&                             ShareConnectivityFromZone)
 CHARACTER*(*)               ZoneTitle
 INTEGER*4                   ZoneType
 INTEGER*4                   IMxOrNumPts
 INTEGER*4                   JMxOrNumElements
 INTEGER*4                   KMxOrNumFaces
 INTEGER*4                   ICellMax
 INTEGER*4                   JCellMax
 INTEGER*4                   KCellMax
 DOUBLE PRECISION            Solution Time
 INTEGER*4                   StrandID
 INTEGER*4                   ParentZone
 INTEGER*4                   IsBlock
 INTEGER*4                   NumFaceConnections
 INTEGER*4                   FaceNeighborMode
 INTEGER*4                   TotalNumFaceNodes,
 INTEGER*4                   NumConnectedBoundaryFaces
 INTEGER*4                   TotalNumBoundaryConnections
 INTEGER*4                   PassiveVarList(*)
 INTEGER*4                   ValueLocation(*)
 INTEGER*4                   ShareVarFromZone(*)
 INTEGER*4                   ShareConnectivityFromZone
```

C Syntax:

```
#include TECIO.h
```

```
##INTEGER4  TECZNE142(char  *ZoneTitle,
                      INTEGER4 *ZoneType,
                      INTEGER4 *IMxOrNumPts,
                      INTEGER4 *JMxOrNumElements,
                      INTEGER4 *KMxOrNumFaces,
                      INTEGER4 *ICellMax,
                      INTEGER4 *JCellMax,
                      INTEGER4 *KCellMax,
                      double  *SolutionTime,
                      INTEGER4 *StrandID,
                      INTEGER4 *ParentZone,
                      INTEGER4 *IsBlock,
                      INTEGER4 *NumFaceConnections,
                      INTEGER4 *FaceNeighborMode,
                      INTEGER4 *TotalNumFaceNodes,
                      INTEGER4 *NumConnectedBoundaryFaces,
                      INTEGER4 *TotalNumBoundaryConnections,
                      INTEGER4 *PassiveVarList,
                      INTEGER4 *ValueLocation,
                      INTEGER4 *ShareVarFromZone,
                      INTEGER4 *ShareConnectivityFromZone)
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Applies to Zone Type(s)	Notes
ZoneTitle	ALL	The title of the zone. Must be null-terminated.
ZoneType	ALL	The type of the zone: 0=ORDERED 1=FELINESEG 2=FETRIANGLE 3=FEQUADRILATERAL 4=FETETRAHEDRON 5=FEBRICK 6=FEPOLYGON 7=FEPOLYHEDRON
IMax or NumPts	ALL	For ordered zones, the number of nodes in the I-index direction. For finite element zones (cell-based and face-based), the number of nodes.

Parameter	Applies to Zone Type(s)	Notes
JMax or NumElements	ALL	For ordered zones, the number of nodes in the J index direction. For finite element zones (cell-based and face-based), the number of elements.
KMax or NumFaces	ORDERED FEPOLYGON FEPOLYHEDRON	For ordered zones, the number of nodes in the K index direction. For polyhedral and polygonal finite element zones, the number of faces. Not used for all other finite element zone types.
ICellMax	N/A	Reserved for future use. Set to zero.
JCellMax	N/A	Reserved for future use. Set to zero.
KCellMax	N/A	Reserved for future use. Set to zero.
SolutionTime	ALL	Scalar double precision value specifying the time associated with the zone. Refer to User's Manual for additional information on working with transient data.
StrandID	ALL	Scalar integer value specifying the strand to which the zone is associated. 0 = static zone, not associated with a strand. Values greater than 0 indicate a zone is assigned to a given strand. Refer to User's Manual for additional information on strands. If you are converting your function calls from function calls 109 or older, use "0" for StrandID.

Parameter	Applies to Zone Type(s)	Notes
ParentZone	ALL	ParentZone is no longer used. Enter 0 for this value.
IsBlock	ALL	Deprecated field. Always set to 1.
NumFaceConnections	ORDERED FELINESEG FETRIANGLE FEQUADRILATERAL FETETRAHEDRON FEBRICK	Used for cell-based finite element and ordered zones only. The number of face connections that will be passed in routine TECFACE142.
FaceNeighborMode		Used for cell-based[1] finite element and ordered zones only. The type of face connections that will be passed in routine TECFACE142. 0=LocalOneToOne 1=LocalOneToMany 2=GlobalOneToOne 3=GlobalOneToMany
TotalNumFaceNodes	FEPOLYGON FEPOLYHEDRON	Used for face-based[2] finite element zones Total number of nodes for all faces. It is also the sum of the FaceNodeCounts array (defined in TECPOLYFACE142). For polygonal zones this value is equivalent to 2 * NumFaces. NumFaces = the number of faces in the zone. Refer to FaceNodeCounts and FaceNodes for simple example.
NumConnectedBoundaryFaces	FEPOLYGON FEPOLYHEDRON	Used for face-based[2] finite element zones. Total number of boundary faces, where boundary faces are faces that either have more than one neighboring cell on a side or have a neighboring cell from another zone. Refer to Boundary Faces and Boundary Connections for simple example.

Parameter	Applies to Zone Type(s)	Notes
TotalNumBoundaryConnections	FEPOLYGON FEPOLYHEDRON	Used for face-based[2] finite element zones. Total number of boundary connections for all faces. In general, TotalNumBoundaryConnections will be equal to NumConnectedBoundaryFaces However, TotalNumBoundaryConnections must be greater than or equal to equal to NumConnectedBoundaryFaces. Refer to Boundary Faces and Boundary Connections for simple example.
PassiveVarList	ALL	Array, dimensioned by the number of variables, of 4 byte integer values specifying the active/passive nature of each variable. A value of 0 indicates the associated variable is active while a value of 1 indicates that it is passive. If all variables are active, you may pass NULL rather than an array of zeroes. Refer to [introduction/best-practices/passive-variables] for additional information.
ValueLocation	ALL	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-centered. Pass null to indicate that all variables are node-centered.

Parameter	Applies to Zone Type(s)	Notes
ShareVarFromZone	ALL	Indicates variable sharing. Array, dimensioned by the number 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 TECDAT142. 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	ALL	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. NOTE: Connectivity and/or face neighbors cannot be shared when the face neighbor mode is set to Global. Connectivity cannot be shared between cell-based and face-based finite element zones.

[1] Cell-based finite element zones: FELINESEG, FETRIANGLE, FEQUADRILATERAL, FETETRAHEDRON, and FEBRICK.
[2] Face-based finite element zones: FEPOLYGON and FEPOLYHEDRON. For these consider using TECPOLYZNE142 instead.

Examples:

Refer to the following examples for illustrations of working with TECZNE142:

- Face Neighbors
- Writing to Multiple Binary Data Files
- Polygonal Example
- Multiple Polyhedral Zones
- Multiple Polygonal Zones

- Polyhedral Example
- Partitioned Data Examples

TECZNEFEMIXED142

Writes header information about the next zone to be added to the data file. TECZNEFEMIXED142 specifies that the next zone is a finite element zone with one or more linear or high order sections. A mixed element zone can have between 1 and 16 sections. All cells within a section have the same cell type, grid order, and basis function. The cell types of all sections within a zone must have the same spatial dimensionality, or in other words, all sections of a zone must either be all line, surface, or volume cell types. When using TecIO-MPI, TECZNEFEMIXED142 should be immediately followed by a call to TECZNEMAP142. After TECZNEFEMIXED142 and TECZNEMAP142 are called, you must call TECDAT142 one or more times and then TECNOD142 (after calling TECDAT142).

FORTRAN Syntax:

```
 INTEGER*4 FUNCTION TECZNEFEMIXED142(ZoneTitle,
&                                    NumNodes,
&                                    NumSections,
&                                    CellShapePerSection,
&                                    GridOrderPerSection,
&                                    BasisFnPerSection,
&                                    NumElementsPerSection,
&                                    SolutionTime,
&                                    StrandID,
&                                    NumFaceConnections,
&                                    FaceNeighborMode,
&                                    PassiveVarList,
&                                    ValueLocation,
&                                    ShareVarFromZone,
&                                    ShareConnectivityFromZone)
 CHARACTER*(*)                       ZoneTitle
 INTEGER*8                           NumNodes
 INTEGER*4                           NumSections
 INTEGER*4                           CellShapePerSection
 INTEGER*4                           GridOrderPerSection
 INTEGER*4                           BasisFnPerSection
 INTEGER*8                           NumElementsPerSection
 DOUBLE PRECISION                    SolutionTime
 INTEGER*4                           StrandID
 INTEGER*4                           NumFaceConnections
 INTEGER*4                           FaceNeighborMode
 INTEGER*4                           PassiveVarList
 INTEGER*4                           ValueLocation
 INTEGER*4                           ShareVarFromZone
```

```
 INTEGER*4                          ShareConnectivityFromZone
```

C Syntax:

```
#include "TECIO.h"
INTEGER4 TECZNEFEMIXED142(char const*  ZoneTitle,
                          INTEGER8 const* NumNodes,
                          INTEGER4 const* NumSections,
                          INTEGER4 const* CellShapePerSection,
                          INTEGER4 const* GridOrderPerSection,
                          INTEGER4 const* BasisFnPerSection,
                          INTEGER8 const* NumElementsPerSection,
                          double  const* SolutionTime,
                          INTEGER4 const* StrandID,
                          INTEGER4 const* NumFaceConnections,
                          INTEGER4 const* FaceNeighborMode,
                          INTEGER4 const* PassiveVarList,
                          INTEGER4 const* ValueLocation,
                          INTEGER4 const* ShareVarFromZone,
                          INTEGER4 const* ShareConnectivityFromZone)
```

Return Value

0 if successful, -1 if unsuccessful.

Parameters

Parameter	Description
ZoneTitle	The title of the zone. Must be null-terminated.
NumNodes	Total number of nodes for the zone.
NumSections	Number of FE mixed element sections for the zone. Must be between 1 and 16, inclusive.
CellShapePerSection	Array containing the cell shape for each section. Must be between 0 and 6, inclusive. 0=FECellShape_Bar 1=FECellShape_Triangle 2=FECellShape_Quadrilateral 3=FECellShape_Tetrahedron 4=FECellShape_Hexahedron 5=FECellShape_Pyramid 6=FECellShape_Prism
GridOrderPerSection	Array containing the grid order for each section. Grid orders must be between 1 and 4, inclusive.
BasisFnPerSection	Array containing the basis function for each section. Must be 0 for each section.

Parameter	Description
SolutionTime	Scalar double precision value specifying the time associated with the zone. Refer to User's Manual for additional information on working with transient data.
StrandID	Scalar integer value specifying the strand to which the zone is associated. 0 = static zone, not associated with a strand. Values greater than 0 indicate a zone is assigned to a given strand. Refer to User's Manual for additional information on strands. If you are converting your function calls from function calls 109 or older, use "0" for StrandID.
NumFaceConnections	The number of face connections that will be passed in routine TECFACE142.
FaceNeighborMode	The type of face connections that will be passed in routine TECFACE142. 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 indicates the associated variable is active while a value of 1 indicates that it is passive. If all variables are active, you may pass NULL rather than an array of zeroes. Refer to [introduction/best-practices/passive-variables] for additional information.
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-centered. Pass null to indicate that all variables are node-centered.
ShareVarFromZone	Indicates variable sharing. Array, dimensioned by the number 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 TECDAT142. 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	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. NOTE: Connectivity and/or face neighbors cannot be shared when the face neighbor mode is set to Global. Connectivity cannot be shared between cell-based and face-based finite element zones.

TECZNEMAP142

When using TecIO-MPI, must be called immediately after calling TECZNE142, or TECZNEFEMIXED142 and must be called by the main process and by each process that will write a zone to indicate which

processes will output each partition of the zone. A zone may be written in non-partitioned fashion (that is, as in the standard TecIO library) by indicating that a single process will output it (`npartitions` of 1, and a one-element `ptnranks` array). When outputting a partitioned zone, each rank outputting this zone should then call `TECFEPTN142` or `TECIJKPTN142` followed by data output calls for each partition it will output.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECZNEMAP142(npartitions, ptnranks)
INTEGER*4          npartitions
INTEGER*4          ptnranks
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECZNEMAP142(INTEGER4 *npartitions,
                      INTEGER4 *ptnranks);
```

Return Value

0 if successful, nonzero if unsuccessful.

Parameters

Parameter	Description
npartitions	The number of partitions for this zone (1 for a non-partitioned zone).
ptnranks	An array of MPI ranks (processes) that indicates the process that will output that partition. The array may include the main process. For non-partitioned zones, contains a single entry.

Defining Polyhedral and Polygonal Data

Polyhedral data is defined using TECPOLYZNE142 (or TECZNE142), TECPOLYFACE142, and TECPOLYBCONN142. Via TECPOLYZNE142 the number of nodes, faces, elements, boundary faces, and boundary connections are specified. TECPOLYFACE142 is used to specify the face mapping. If the zone is connected to neighboring zones, TECPOLYBCONN142 is then used to specify those connections.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Before defining your polyhedral or polygonal data, you should determine the numbering scheme for the nodes, faces and elements in each zone of your data set. The numbering scheme is communicated to Tecplot implicitly by the order in which you supply the data. For example, the first nodal value

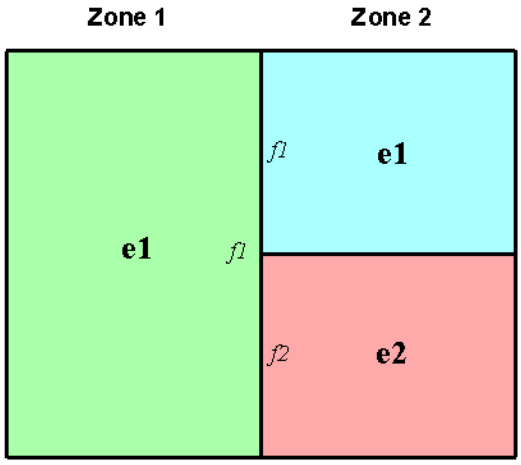
supplied is for Node 1, followed by the value for Node 2, continuing to node N (where N is the total number of nodes). Similarly, for faces and elements.

The remainder of this section provides simple examples illustrating how to define polygonal and polyhedral data.

Boundary Faces and Boundary Connections

A "Connected Boundary Face" is a face with at least one neighboring element that belongs to another zone. Each "Connected Boundary Face" has one or more "Boundary Connections". A "Boundary Connection" is defined as the element-zone tuple used to identify the neighboring element when the element is part of another zone.

Consider the following picture:



In the figure shown above, Zone 1 contains a single element (e1) and Zone 2 contains two elements (e1 and e2). The boundary faces and boundary connections for each zone are as follows:

Zone 1

In Zone 1, Face 1 (f1) is the sole connected boundary face. It has two boundary connections. The first boundary connection is Element 1 in Zone 2. The second boundary connection is Element 2 in Zone 2.

- NumConnectedBoundaryFaces = 1
- TotalNumBndryConnections = 2

Zone 2

In Zone 2, both Face 1 and Face 2 are connected boundary faces. There is a total of two boundary connections. The boundary connection for each boundary face in Zone 2 is Element 1 in Zone 1.

- NumConnectedBoundaryFaces = 2
- TotalNumBndryConnections = 2

FaceNodeCounts and FaceNodes

For illustration purposes, consider a zone composed of a single pyramidal element. The pyramid is composed of five nodes and five faces.

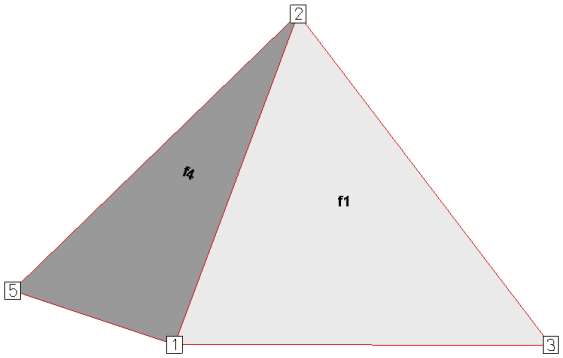


Figure 11. A simple pyramid. The remaining triangular faces are Faces 2 and 3. The bottom rectangular face is Face 5. Node 4 is obscured from view.

The FaceNodeCounts array is used to specify the number of nodes that compose each face. The values in the array are arranged as follows:

```
FaceNodeCounts = [NumNodesInFace1,
                  NumNodesInFace2,
                  ...
                  NumNodesInFaceF]
```

where F is the total number of faces in the zone.

In this example, the FaceNodeCounts array is: [3 3 3 3 4]. The first four faces are composed of three nodes and the last face is composed of four nodes.

The FaceNodes array is used to specify which nodes belong to which face. The array is dimensioned by the total number of face nodes in the zone (specified via `TECPOLYZNE142`. The total number of face nodes is defined as the sum of the number of nodes in each face.

The first K values in the FaceNodes array are the node numbers for Face 1, where K is the first value in the FaceNodeCounts array. The next L values are the node numbers for Face 2, where L is the second value in the FaceNodeCounts array.

When supplying the node numbers for each face, you must supply the numbers in either a clockwise or counter-clockwise configuration around the face. Otherwise, the faces will be contorted when the data is plotted.

It is not important to be consistent when choosing between clockwise or counter-clockwise ordering. The key is to present the numbers consistently within the numbering scheme. For example, you may present the node numbers for face 1 in a

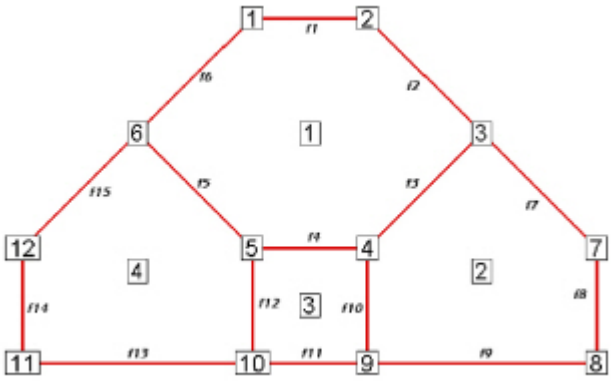
clockwise order and the node numbers for the remaining faces in counter-clockwise order.

Consider the pyramid used above. Using the FaceNodeCounts array we have already defined and the figure, we can create the FaceNodes array for the pyramid.

```
FaceNodes = [1, 2, 3
             3, 2, 4,
             5, 2, 4,
             5, 1, 2,
             1, 5, 4, 3]
```

FaceRightElems and FaceLeftElems

After specifying the face map data (using the FaceNodeCounts and FaceNodes array), the next step is to identify the element on either side of each face. To illustrate this, we will switch from the single element zone to the following data set:



The neighboring elements can be determined using the right-hand rule:

2D Data

For each face, place your right-hand along the face with your fingers pointing in the direction of incrementing node numbers (i.e. from Node 1 to Node 2). The right side of your hand will indicate the right element, and the left side of your hand will indicate the left element.

3D Data

For each face, curl the fingers of your right-hand following the order that the nodes were presented in the FaceNodes array. Your thumb will point to the right element. The left element is the other adjacent element. If the face has more than one neighboring element on a single side, you will need to use the FaceBoundaryConnectionCounts, FaceBoundaryConnectionElems and FaceBoundaryConnectionZones array.

The neighboring elements for each face are stored in the FaceRightElems and FaceLeftElems array.

Each array is dimensioned by the total number of faces in the zone. The first value in each array is the right or left neighboring element for Face 1, followed by the neighboring element for Face 2, and so forth.

```
FaceRightElems = [RightNeighborToFace1,
                  RightNeighborToFace2,
                  ...
                  RightNeighborToFaceF]
```

```
FaceLeftElems = [LeftNeighborToFace1,
                 LeftNeighborToFace2,
                 ...
                 LeftNeighborToFaceF]
```

where F is the total number of faces

In the above plot, the face neighbors are as follows:

Face Number	**Right Neighboring Element**	**Left Neighboring Element**
Face 1	1	0
Face 2	1	0
Face 3	1	2
Face 4	1	3
Face 5	1	4
Face 6	1	0
Face 7	2	0
Face 8	2	0
Face 9	2	0
Face 10	2	3
Face 11	3	0
Face 12	3	4
Face 13	4	0
Face 14	4	0

Face Number	Right Neighboring Element	Left Neighboring Element
Face 15	4	0

The number zero is used to indicate that the face is on the edge of the data (i.e. has "no neighboring element").

FaceBoundaryConnectionElements and Zones

When working with multiple zones, an additional aspect is folded into the FaceLeftElems and FaceRightElems arrays. When the neighboring element is not within the current zone, you cannot identify the element by its element number alone. Instead you need to specify both the element number and its zone number. This is accomplished using the FaceBoundaryConnectionElements and FaceBoundaryConnectionZones arrays. For each boundary connection, the element number of the boundary connection is stored in the FaceBoundaryConnectionElements array while its zone number is stored in the FaceBoundaryConnectionZones array.

A negative value in the FaceLeftElems or FaceRightElems array is used to indicate that the neighboring element belongs to another zone. The magnitude of the negative number is a pointer to a value in the FaceBoundaryConnectionElements and FaceBoundaryConnectionZones arrays. For example, given the following FaceBoundaryConnectionElements and FaceBoundaryConnectionZones arrays:

```
FaceBoundaryConnectionElements = [ 1 1 3 4 ]
```

```
FaceBoundaryConnectionZones = [ 2 2 3 3 ]
```

A value of -4 in the FaceLeftElems indicates that the left neighboring element for that face is element four in zone three.

Partially Obscured Boundary Faces

A face on the boundary of a zone may be partially obscured by its boundary connections (neighboring elements). While Tecplot 360 does not draw fully obscured boundary faces (because it treats those faces as internal faces), Tecplot 360 does draw partially obscured boundary faces. Thus, Tecplot 360 requires definition of partially obscured boundary faces.

To indicate a partially obscured face, indicate the appropriate neighboring element as zero in the FaceBndryConnectionElems and FaceBndryConnectionZones arrays, followed by the actual neighboring elements. When Tecplot 360 sees a list of neighboring elements for a boundary face that begin with element zero, it marks that boundary face as partially obscured.

If Tecplot 360 sees a zero in FaceBndryConnectionElems that is not the first boundary element listed for a face, an error message will appear, indicating that either the partially obscured boundary face

was not indicated correctly, or FaceBndryConnectionsElems and/or FaceBndryConnectionsZones was not completely filled out.

Examples

Source code for example programs that use the TecIO library is provided with your Tecplot 360 installation in the `util/tecio/examples` folder in the installation directory. See the `readme.txt` file in that folder for additional information on building the examples.

The examples (written in C) provide a basic illustration of creating a `*.plt` file using the TecIO library. If you plan to compile the examples, be sure to review the instructions in Linking with the TecIO Library first.

In order to keep the examples as simple as possible, error checking is not included. For complete details on the parameters used and the function syntax for each TecIO function, refer to Binary Data File Function Reference. When creating a binary data file using the TecIO library, the functions must be called in a specific order. Refer to Binary Data File Function Calling Sequence for details.

Face Neighbors

This example illustrates how to (1) create two simple FE-quadrilateral zones and (2) create a face neighbor connection between the two zones. In order to keep the example as simple as possible, error checking is not included. If you plan to compile this example, be sure to include `TECIO.h`.

For complete details on the parameters used and the function syntax for each TecIO function, refer to Binary Data File Function Reference. When creating a binary data file using the TecIO library, the functions must be called in a specific order. Refer to Binary Data File Function Calling Sequence for details.

Step 1 Initialize the data file using TECINI

`TECINI` is required for all data files. It is used to open the data file and initialize the file header information (name the data file, the variables for the data file, and the file type).

```
INTEGER4 Debug  = 1;
INTEGER4 VIsDouble  = 0;
INTEGER4 FileType  = 0;
INTEGER4 FileFormat = 0; // 0 == PLT, 1 == SZPLT
INTEGER4 I  = 0;  /* Used to track return codes */

I = TECINI142((char*)"Face Neighbors Example", /* Specifies the name
                                                * of the entire
                                                * dataset
                                                */
              (char*)"X Y P",                 /* Defines the
                                                * variables for the
```

```
                                        * data file.  Each
                                        * zone must contain
                                        * each of the vars
                                        * listed.  The order
                                        * of the variables in
                                        * the list is used to
                                        * define the variable
                                        * number (e.g. X is
                                        * Var 1.)
                                        */
            (char*)"FaceNeighbors.plt",  /* Specifies the
                                        * file name.
                                        */
            (char*)".",
            &FileFormat,
            &FileType,                  /* The FileType is set to
                                        * zero, indicating it is
                                        * a full file containing
                                        * both grid and solution
                                        * data).
                                        */
            &Debug,
            &VIsDouble);
```

Step 2 Create Zone 1

After TECINI is called, call `TECZNE to create one or more zones for your data file.

```
INTEGER4 ZoneType  = 3;  /* set the zone type to
                          * FEQuadrilateral
                          */
INTEGER4 NumPts    = 6;
INTEGER4 NumElems  = 2;
INTEGER4 NumFaces  = 8;
INTEGER4 ICellMax  = 0;  /* not used */
INTEGER4 JCellMax  = 0;  /* not used */
INTEGER4 KCellMax  = 0;  /* not used */
double  SolTime    = 360.0;
INTEGER4 StrandID  = 0;  /* StaticZone */
INTEGER4 unused    = 0;  // ParentZone is no longer used
INTEGER4 IsBlock   = 1;  /* Block */
INTEGER4 NFConns   = 1;  /* Specify the number of Face
                          * Neighbor Connections in the
                          * Zone.  When this value is
                          * greater than zero, TECFACE must
                          * be called prior to creating the
```

```
                         * next zone or ending the file.
                         */

/* Specify the Face Neighbor Mode.
 * A value of 2 indicated that the face neighbor mode is global
 * one-to-one.  The scope of the face neighbors (local or
 * global) is with respect to the zones. A value of global
 * indicates that the face neighbor(s) is/are shared aross zones;
 * a value of local indicates that the face neighbor(s) are
 * shared within the current zone.  The terms one-to-one and
 * one-to-many are used to indicate whether the face in question
 * is shared with one cell or several cells.
 * For example, if your data is arranged as follows:
 *
 *             +-----+-----+-----+
 *             |     |     |     |
 *             |  1  |  2  |  3  |
 *             |     |     |     |
 *             +-----+-----+-----+
 *             |     |           |
 *             |  4  |     5     |
 *             |     |           |
 *             +-----+-----------+
 *
 * The face between 1 & 4 is local-one-to-one. The face between
 * 5 and (2 & 3) is local one-to-many.
 */

INTEGER4 FNMode  = 2;
INTEGER4 TotalNumFaceNodes  = 1;           /* Not used for FEQuad zones*/
INTEGER4 NumConnectedBoundaryFaces  = 1;   /* Not used for FEQuad zones*/
INTEGER4 TotalNumBoundaryConnections  = 1; /* Not used for FEQuad zones*/
INTEGER4 ShrConn  = 0;

INTEGER4 ValueLocation[3] = {1, 1, 1};  /* Specify the variable
                                         * values at the nodes.
                                         * NOTE:  Because all of
                                         * the variables are
                                         * defined at the nodes,
                                         * we can just pass
                                         * NULL for this array.
                                         * We are providing the
                                         * array for illustration
                                         * purposes.
                                         */

I = TECZNE142((char*)"Zone 1",
              &ZoneType,
```

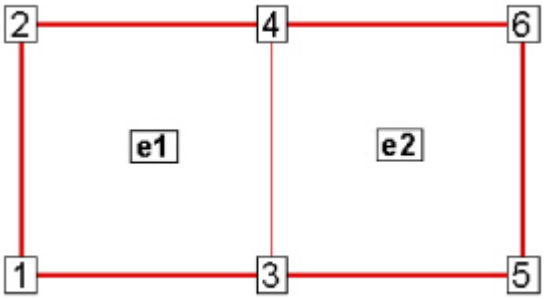
```
                &NumPts,
                &NumElems,
                &NumFaces,
                &ICellMax,
                &JCellMax,
                &KCellMax,
                &SolTime,
                &StrandID,
                &unused,
                &IsBlock,
                &NFConns,
                &FNMode,
                &TotalNumFaceNodes,
                &NumConnectedBoundaryFaces,
                &TotalNumBoundaryConnections,
                NULL,
                ValueLocation,
                NULL,
                &ShrConn);
```

Step 3 Define the node numbering for Zone 1

For this example, we will create 2 rectangular cells in Zone 1. Before defining your variables, you must establish a consistent node numbering scheme for your data. Once the node numbers are defined, supply the variable values in the node numbering order. In this example, Node 1 is defined at `X = 0` and `Y = 0`. As such, the first value supplied for X (i.e. `X[0]`) is 0. Similarly, the first value supplied for Y is 0.

It is important that you refer to node numbers consistently. The node numbers will be used later to define the connectivity for each element.

For this example, we will create two quadrilateral elements. The node numbering for the elements is defined in the following picture.



Step 4 Set up the variable values

The variable values will be written to the file using TECDAT. Because we are specifying nodal variables (as specified via the ValueLocation parameter in TECZNE), each variable is dimensioned by the number of points (NumPts) in the Zone. You have the option to specify some variables with nodal values and some with cell-centered values. Refer to TECZNE142 or TECZNEFEMIXED142 for details.

```
float *X = new float[NumPts];
float *Y = new float[NumPts];
float *P = new float[NumPts];

/* For this example, we will create 2 rectangular cells in Zone
 * 1.  Before defining your variables, you must establish a
 * consistent node numbering scheme for your data.  Once the
 * node numbers are defined, supply the variable values in the
 * node numbering order.  In this example, node 1 is defined at
 * X = 0 and Y = 0.  As such, the first value supplied for X
 * (i.e. X[0]) is 0.  Similarly, the first value supplied for Y
 * is 0.
 *
 * It is important that you refer to node numbers consistently.
 * The node numbers will be used later to define the
 * connectivity for each element.
 */

X[0] = 0;
X[1] = 0;
X[2] = 1;
X[3] = 1;
X[4] = 2;
X[5] = 2;

Y[0] = 0;
Y[1] = 1;
Y[2] = 0;
Y[3] = 1;
Y[4] = 0;
Y[5] = 1;

for (INTEGER4 ii = 0; ii < NumPts; ii++)
 P[ii] = (float)(NumPts - ii);

INTEGER4 DIsDouble =  0;  /* Set DIsDouble to zero to use
                           * variables in float format.
                           */

/* Call TECDAT once for each variable */
```

```
I  = TECDAT142(&NumPts, &X[0], &DIsDouble);
I  = TECDAT142(&NumPts, &Y[0], &DIsDouble);
I  = TECDAT142(&NumPts, &P[0], &DIsDouble);
```

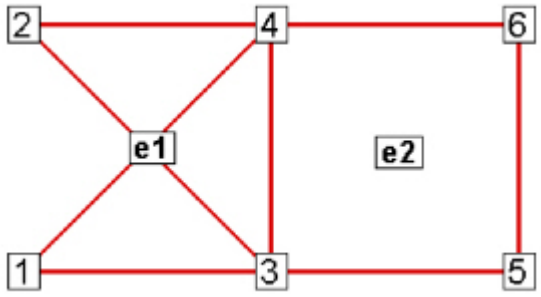
Step 5 Define the connectivity list for Zone 1

The Connectivity List is used to specify the nodes that compose each element. When working with nodal variables, the numbering of the nodes is implicitly defined when the variables are declared. The first value of each variable is for node one, the second value for node two, and so on.

Because this zone contains two quadrilateral elements, we must supply 8 values in the connectivity list. The first four values define the nodes that form Element 1. Similarly, the second four values define the nodes that form Element 2.

```
INTEGER4 ConnList[8] = {1, 3, 4, 2,
                        3, 5, 6, 4 };
I  = TECNOD142(ConnList);
```

It is important to provide the node list in either a clockwise or counter-clockwise order. Otherwise, your elements will be misshapen. For example, if the first two numbers in the above connectivity list were switched, the zone would appear as follows:



Step 6 Define the face neighbor connections for Zone 1

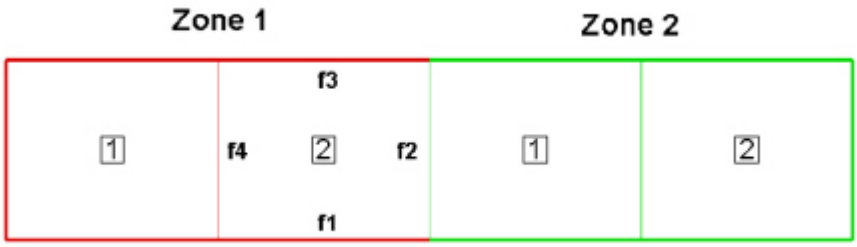
Now that TECNOD or TECNODE has been called, the creation of Zone 1 is complete. However, in this example, we will define a face neighbor between Zone 1 and Zone 2 (to be created later in the example). Face Neighbor connections are used to define connections that are not created via the connectivity list. For example, local face neighbors may need to be defined when a zone wraps onto itself and global face neighbors may need to be defined to smooth edges across zones. Face Neighbors are used when deriving variables and drawing contours.

In this example, we are creating a face neighbor connection between Cell 2 in Zone 1 and Cell 1 in Zone 2. The information required when specifying face neighbors depends upon the type of connection. Refer to TECFACE142 for details.

In this case, we must supply the following information (in the order provided):

- the cell number in the current zone that contains the face neighbor
- the number of the face in that cell that contains the face neighbor
- the number of the other zone to which the face is connected
- the number of the cell in the other zone to which the face is connected

In this example, Face 2 in Cell 2 in the current zone is connected to Cell 1 in Zone 2.




```
INTEGER4 FaceConn[4] = {2, 2, 2, 1};
I  = TECFACE142(FaceConn);
```

Step 7 Create Zone 2

The creation of Zone 1 is complete. We are ready to create Zone 2. For simplicity, Zone 2 is a copy of Zone 1 shifted along the X-axis. As such, many of the variables used to create Zone 1 are re-used here.

```
/* Call TECZNE to create Zone 2 */
I = TECZNE142((char*)"Zone 2",
              &ZoneType,
              &NumPts,
              &NumElems,
              &NumFaces,
              &ICellMax,
              &JCellMax,
              &KCellMax,
              &SolTime,
              &StrandID,
              &unused,
              &IsBlock,
              &NFConns,
              &FNMode,
              &TotalNumFaceNodes,
              &NumConnectedBoundaryFaces,
              &TotalNumBoundaryConnections,
              NULL,
```

```
              ValueLocation,
              NULL,
              &ShrConn);
```

Step 8 Define the variables for Zone 2

Because Zone 2 is a copy of Zone 1, shifted along the X-axis, we can share the Y variable definition used to Zone. We will also create a second pressure variable for Zone 2 (P2).

```
float *X2 = new float[NumPts];
float *P2 = new float[NumPts];

for (INTEGER4 ii = 0; ii < NumPts; ii++)
{
 X2[ii] = X[ii] + 2;
 P2[ii] = 2 * (float)ii;
}

I  = TECDAT142(&NumPts, &X2[0], &DIsDouble);
I  = TECDAT142(&NumPts, &Y[0], &DIsDouble);
I  = TECDAT142(&NumPts, &P2[0], &DIsDouble);

delete X;
delete Y;
delete P;
delete X2;
delete P2;
```

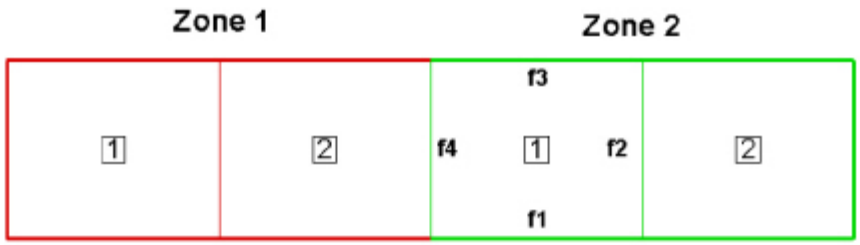
Step 9 Define the connectivity list for Zone 2

As with Zone 1, we must define the connectivity list for Zone 2. Because, the node numbering restarts at one for each new zone and the nodal arrangement is identical between the two zones, we may reuse the connectivity list from Zone 1.

```
I  = TECNOD142(ConnList);
```

Step 10 Define the face neighbor connections for Zone 2

We will now specify the face neighbor connection with respect to our new current zone of Zone 2.




```
INTEGER4 FaceConn2[4] = {1, 4, 1, 2};  /* cell 1, face 4 in
                                        * current zone is a
                                        * neighbor to cell 2 in
                                        * zone 1.
                                        */
I  = TECFACE142(FaceConn2);
```

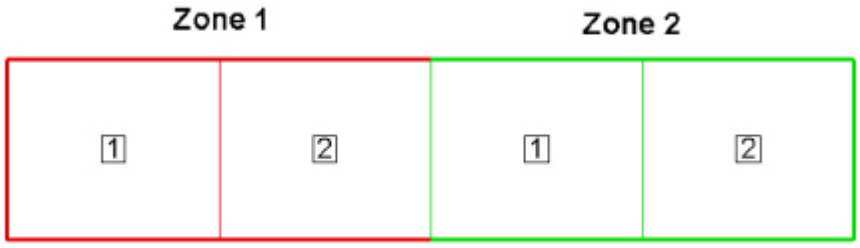
Step 11 Close the file

Call TECEND to close the file.

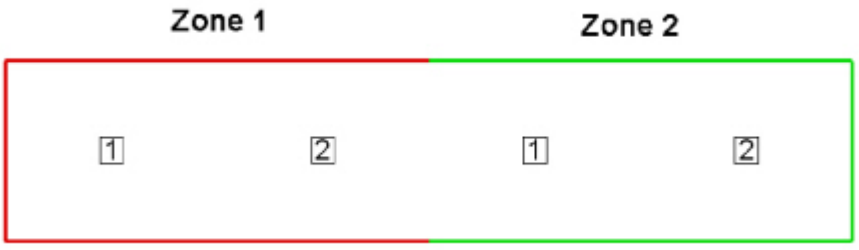
```
I = TECEND142();
```

Summary

When the preceding code is compiled and built, the data file will look as follows (with the Mesh and Edge layers turned-on):



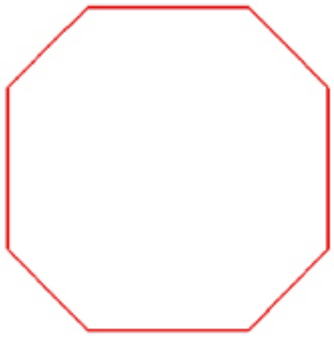
With the Mesh layer deactivated, the data set will look as follows:



If we had not included face neighbor connections, an Edge line would be drawn in between the two zones.

Polygonal Example

The following example (written in C++) illustrates how to create a single octagonal cell using the TecIO library.



In order to keep the example as simple as possible, error checking is not included. If you plan to compile this example, be sure to include `TECIO.h`.

For complete details on the parameters used and the function syntax for each TecIO function, refer to Binary Data File Function Reference. When creating a binary data file using the TecIO library, the functions must be called in a specific order. Refer to Binary Data File Function Calling Sequence for details.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Step 1 Initialize the data file using TECINI

`TECINI` is required for all data files. It is used to: open the data file and initialize the file header information (name the data file, the variables for the data file, and the file type).

```
INTEGER4 Debug  = 1;
INTEGER4 VIsDouble  = 0;
INTEGER4 FileType  = 0;
INTEGER4 FileFormat = 0; // 0 == PLT, 1 == SZPLT; Only PLT is currently
                         // supported for polygonal zones.
INTEGER4 I;              // used to check return codes

/*
 * Open the file and write the Tecplot datafile
 * header information
 */

I = TECINI142((char*)"Octagon",
```

```
              (char*)"X Y P",  /* Defines the variables for the data
                               * file. Each zone must contain each
                               * of the vars listed here. The order
                               * of the variables in the list is
                               * used to define the variable number
                               * (e.g. X is Variable 1). When
                               * referring to variables in other
                               * TecIO functions, you will refer to
                               * thevariable by its number.
                               */
              (char*)"octagon.plt",
              (char*)".",      /* scratch directory */
              &FileFormat,
              &FileType,
              &Debug,
              &VIsDouble);
```

Step 2 Create Zone 1

After TECINI is called, call TECPOLYZNE to create one or more zones for your data file.

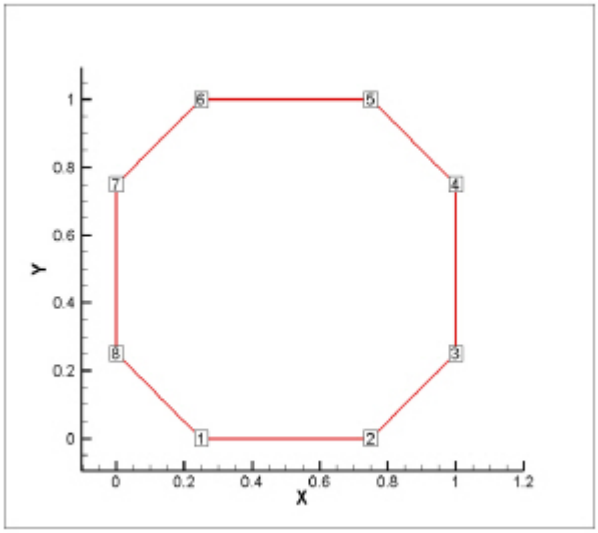
```
/* In this example, we will create a single octagonal cell in
 * Tecplot 360's polyhedral file format.
 */
INTEGER4 ZoneType  = 6;  /* FEPolygon */
INTEGER4 NumNodes  = 8;  /* Number of nodes in the octagon.*/
INTEGER4 NumElems  = 1;  /* Number of octagonal elements. */
INTEGER8 NumFaces  = 8;  /* Number of faces in the octagon.*/
double  SolTime    = 360.0;
INTEGER4 StrandID  = 0;  /* Static Zone */
INTEGER4 unused  = 0;  // ParentZone is no longer used


/* For polygonal zones, the total number of face nodes is equal
 * to twice the number of nodes.  This is because, each face
 * has exactly two nodes.
 */
INTEGER8 NumFaceNodes  = 2 * NumNodes;
/* Boundary Faces and Boundary Connections are not used in this
 * example.
 */
INTEGER4 NumBFaces  = 0;
INTEGER4 NumBConnections = 0;

INTEGER4 ShrConn  = 0;
```

```
I = TECPOLYZNE142((char*)"Octagonal Zone",
                  &ZoneType,
                  &NumNodes,
                  &NumElems,
                  &NumFaces,
                  &NumFaceNodes,
                  &SolTime,
                  &StrandID,
                  &unused,
                  &NumBFaces,
                  &NumBConnections,
                  NULL,
                  NULL,  /* When Value Location is not specified,
                          * Tecplot will treat all variables as
                          * nodal variables.
                          */
                  NULL,
                  &ShrConn);
```

Step 3 Define node numbering

For this example, we will create a single octagonal cell. Before defining your variables, you must establish a consistent node numbering scheme for your data. Once the node numbers are defined, supply the variable values in the node numbering order. In this example, Node 1 is defined at `X = .25` and `Y = 0`. As such, the first value supplied for X (i.e. `X[0]`) is .25. Similarly, the first value supplied for Y is 0.

It is important that you refer to node numbers consistently. The node numbers will be used later to define the connectivity for each element.



Step 4 Set up the variable values

Write the variable values to the file using TECDAT. Because we are specifying nodal variables (as specified via the ValueLocation parameter in TECPOLYZNE), each variable is dimensioned by the number of points (NumPts) in the Zone. You have the option to specify some variables with nodal values and some with cell-centered values. Refer to TECPOLYZNE142 for details.

The order of the values supplied for each nodal variable is determined by the node numbering established in Step 3. The first value for each variable is for Node 1, the second value for each variable is for Node 2 and so forth.

```
V1 = {ValueAtNode1, ValueAtNode2, ⋯, ValueAtNodeN}
```

where N is the total number of nodes

```
float *X = new float[NumNodes];
float *Y = new float[NumNodes];
float *P = new float[NumNodes];

//Define the grid values.
X[0] = 0.25;
Y[0] = 0.0;

X[1] = 0.75;
Y[1] = 0.0;

X[2] = 1.0;
Y[2] = 0.25;

X[3] = 1.0;
Y[3] = 0.75;

X[4] = 0.75;
Y[4] = 1.0;

X[5] = 0.25;
Y[5] = 1.0;

X[6] = 0.0;
Y[6] = 0.75;

X[7] = 0.0;
Y[7] = 0.25;

for (INTEGER4 ii = 0; ii < 8; ii++)
 P[ii] = .5;

/* Write out the field data using TECDAT */
```

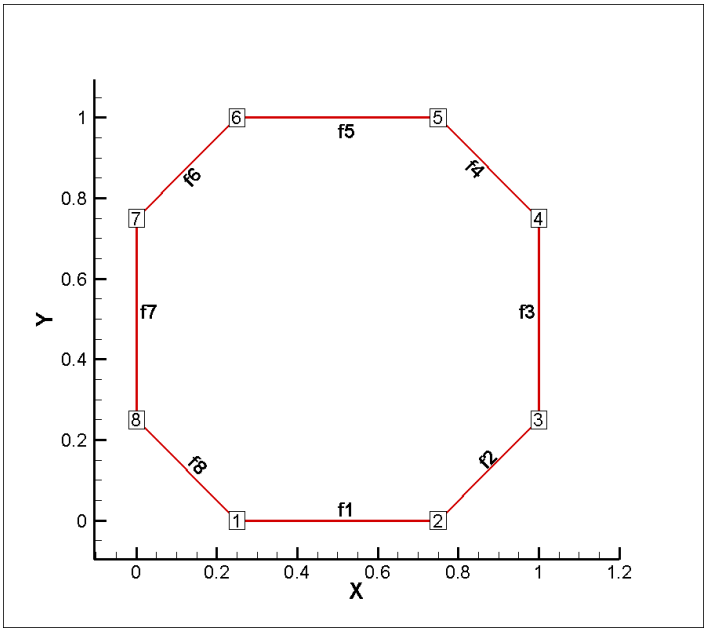
```
INTEGER4 DIsDouble = 0;  /* set IsDouble to 0 to use float
                          * variables. */

I  = TECDAT142(&NumNodes,  X,  &DIsDouble);
I  = TECDAT142(&NumNodes,  Y,  &DIsDouble);
I  = TECDAT142(&NumNodes,  P,  &DIsDouble);

delete X;
delete Y;
delete P;
```

Step 5 Define the Face Nodes

The FaceNodes array is used to indicate which nodes define which face. As mentioned earlier, the number of the nodes is implicitly defined by the order in which the nodal data is provided. The first value of each nodal variable describes Node 1, the second value describes Node 2, and so on.

The face numbering is also implicitly defined. Because there are two nodes in each face of any polygonal zone, the first two nodes provided define Face 1, the next two define Face 2 and so on. If there was a variable number of nodes used to define the faces, the array would be more complicated. Refer to Multiple Polygonal Zones for an example.

The following picture describes the face numbering for this example:



As you can see, Face 1 is defined by Nodes 1 and 2, Face 2 is defined by Nodes 2 and 3, and so forth. Because of this simple arrangement, we can use a for-loop to define all but the end points of the face nodes array.

```
INTEGER4 *FaceNodes = new INTEGER4[NumFaceNodes];
```

```
/*
 * Loop over number of sides, and set each side to two
 * consecutive nodes.
 */
for (INTEGER4 ii = 0; ii < 8; ii++)
{
  FaceNodes[2*ii] = ii + 1;
  FaceNodes[2*ii+1] = ii + 2;
}
FaceNodes[15] = 1;
```

Step 6 Define the right and left elements of each face

The last step for writing out the polygonal data is to define the right and left neighboring elements for each face. The neighboring elements can be determined using the right-hand rule. For each face, place your right-hand along the face with your fingers pointing the direction of incrementing node numbers (i.e. from Node 1 to Node 2). The right side of your hand will indicate the right element, and the left side of your hand will indicate the left element. Refer to FaceRightElems and FaceLeftElems for details.

The number zero is used to indicate that there isn't an element on that side of the face (i.e. the face is on the edge of the data set). This is referred to as "no neighboring element".

Because of the way we numbered the nodes and faces, the right element for every face is the element itself (Element 1) and the left element is "no-neighboring element" (Element 0).

```
INTEGER4 *FaceLeftElems  = new INTEGER4[NumFaces];
INTEGER4 *FaceRightElems = new INTEGER4[NumFaces];

for (INTEGER8 ii = 0; ii < NumFaces; ii++)
{
 FaceLeftElems[ii]  = 0;
 FaceRightElems[ii]  = 1;
}
```

Step 7 Write the face nodes to the file

We can now call TECPOLYFACE142 to write the face nodes to the file. Since we do not have any boundary connections in this data set, there is no need to call TECPOLYBCONN142.

```
INTEGER4 NumFaces32 = (INTEGER4)NumFaces;
I = TECPOLYFACE142(&NumFaces32,
                   NULL,
                   FaceNodes,
                   FaceLeftElems,
                   FaceRightElems);
```

```
delete FaceNodes;
delete FaceLeftElems;
delete FaceRightElems;
```

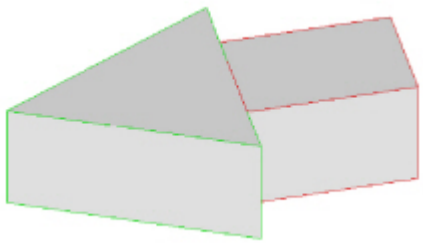
Step 8 Close the file

Call TECEND to close the file.

```
I = TECEND142();
```

Multiple Polyhedral Zones

The following example demonstrates how to create two polyhedral zones, a rectangular solid and a prism. The resulting image is a three-dimensional arrow (shown below).



This example covers the following topics: polyhedral data, working with multiple zones, and specifying partially obscured faces. In order to keep the example as simple as possible, error checking is not included. If you plan to compile this example, be sure to include: TECIO.h.

For complete details on the parameters used and the function syntax for each TecIO function, refer to Binary Data File Function Reference. When creating a binary data file using the TecIO library, the functions must be called in a specific order. Refer to Binary Data File Function Calling Sequence for details.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Step 1 Initialize the data file using TECINI

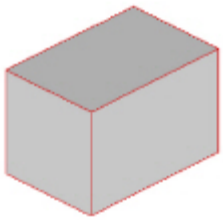
TECINI is required for all data files. This function opens the data file and initializes the file header information (names the data file, the variables for the data file, and the file type).

```
INTEGER4 Debug  = 1;
INTEGER4 VIsDouble = 1;
INTEGER4 FileFormat = 0; // 0 == PLT, 1 == SZPLT; Only PLT is currently
                         // supported for ploy hedral zones
```

```
INTEGER4 FileType  = 0;
INTEGER4 I;

/* Open the file and write the Tecplot datafile
 * header information
 */
I = TECINI142((char*)"Multiple polyhedral zones", /* Name of the entire
                                                   * dataset.
                                                   */
              (char*)"X Y Z P",  /* Defines the variables for the data
                                  * file. Each zone must contain each of
                                  * the variables listed here. The order
                                  * of the variables in the list is used
                                  * to define the variable number (e.g.
                                  * X is Var 1).
                                  */
              (char*)"Arrow.plt",
              (char*)".",         /* Scratch Directory */
              &FileFormat,
              &FileType,
              &Debug,
              &VIsDouble);
```

Step 2 Create Zone 1 (rectangle)

After TECINI is called, call TECPOLYZNE to create one or more zones for your data file. In this example, Zone 1 contains a single rectangular solid created as a face-based finite element (i.e. polyhedral zone). The zone has eight points (or nodes), six faces and one element.



```
/* TECPOLYZNE Parameters */
INTEGER4 ZoneType       = 7; /* sets the zone type to polyhedral */
INTEGER4 NumPts_Rect    = 8;
INTEGER4 NumElems_Rect  = 1;
INTEGER8 NumFaces_Rect  = 6;
double  SolutionTime    = 0.0;
INTEGER4 StrandID       = 0;
INTEGER4 Unused         = 0; // ParentZone is no longer sed
INTEGER4 SharConn       = 0;
```

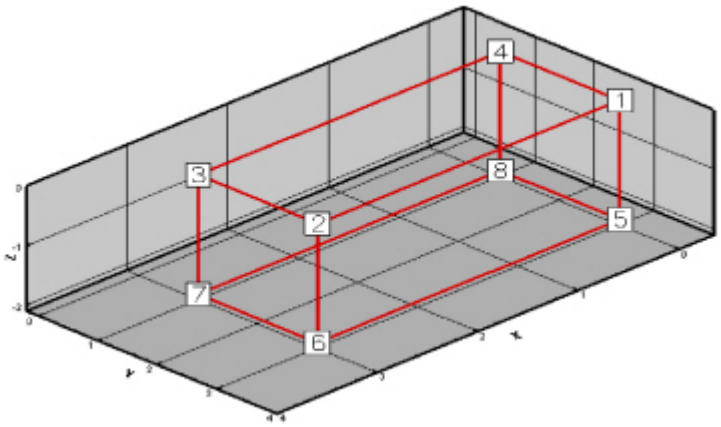
```
/* In a rectangular solid, each face is composed of four nodes.
 * As such, the total number of face nodes is twenty-four (four
 * nodes for each of the six faces).
 */
INTEGER8 TotalNumFaceNodes_Rect  = 24;

/* There is one connected boundary face in this zone (the face on
 * the rectangle adjacent to the arrowhead). Refer to the Data
 * Format Guide for additional information. */
INTEGER4 NumConnBndryFaces_Rect  = 1;

/* The connected boundary face has one connection, the face on
 * the bottom of the arrowhead. A connection is an element-zone
 * tuple that indicates a neighboring element (and its zone) when
 * the neighboring element is in a different zone. Generally,
 * there will be one boundary connection for each boundary face.
 */
INTEGER4 TotalNumBndryConns_Rect  = 1;

/* For illustrative purposes, the grid variables (X, Y, and Z)
 * are nodal variables (i.e. ValueLocation = 1), and the pressure
 * variable (P) is a cell-centered variable (i.e.
 * ValueLocation = 0).
 */
INTEGER4 ValueLocation[4] = { 1, 1, 1, 0 };

I = TECPOLYZNE142((char*)"Zone 1: Rectangular Solid",
                  &ZoneType,
                  &NumPts_Rect,
                  &NumElems_Rect,
                  &NumFaces_Rect,
                  &TotalNumFaceNodes_Rect,
                  &SolutionTime,
                  &StrandID,
                  &Unused,
                  &NumConnBndryFaces_Rect,
                  &TotalNumBndryConns_Rect,
                  NULL,
                  ValueLocation,
                  NULL,
                  &SharConn);
```

Step 3 Set variable values for Zone 1 (rectangle)

Now that the zone has been created, write the variable values to the file by calling TECDAT. While there are more elegant ways to define the grid coordinates for the rectangle, the values are defined explicitly

for simplicity.

Using the picture below, define the variable values.



For nodal variables, provide the values for each variable in nodal order. Similarly, for cell-centered values, provide the variable values in cell order. The location of each variable is specified with TECPOLYZNE.

```
//set variable values (X_Rect, Y_Rect, Z_Rect & P_Rect)
double *X_Rect = new double[NumPts_Rect];
double *Y_Rect = new double[NumPts_Rect];
double *Z_Rect = new double[NumPts_Rect];
double *P_Rect = new double[NumElems_Rect];

for (INTEGER4 ii = 0; ii <= NumPts_Rect / 2; ii += 4)
{
    X_Rect[ii]   = 0;
    X_Rect[ii+1] = 3;
    X_Rect[ii+2] = 3;
    X_Rect[ii+3] = 0;

    Y_Rect[ii]   = 3;
    Y_Rect[ii+1] = 3;
    Y_Rect[ii+2] = 1;
    Y_Rect[ii+3] = 1;
}

for (INTEGER4 ii = 0; ii < 4; ii++)
    Z_Rect[ii] = 0;

for (INTEGER4 ii = 4; ii < NumPts_Rect; ii++)
    Z_Rect[ii] = -2;

P_Rect[0] = 10;
```

```
INTEGER4 IsDouble = 1;
I = TECDAT142(&NumPts_Rect,  X_Rect, &IsDouble);
I = TECDAT142(&NumPts_Rect,  Y_Rect, &IsDouble);
I = TECDAT142(&NumPts_Rect,  Z_Rect, &IsDouble);
I = TECDAT142(&NumElems_Rect, P_Rect, &IsDouble);
```

Step 4 Define the facemap data for Zone 1

Using the following figure, specify which nodes define which face.

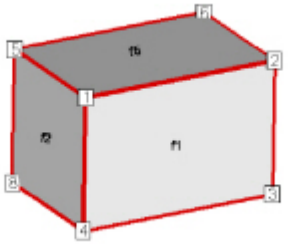


Figure 12. Zone 2 of the sample data. Node 7 is obscured from view and located in the back-left hand corner. Face 6 is the bottom face. Face 3 is opposite Face 1 and Face 4 is opposite Face 2.

In order to specify the face map data, you must first specify how many nodes are in each face using the FaceNodeCounts array. After defining the FaceNodeCounts array, use the FaceNodes array to identify the nodes that compose each face. Refer to FaceNodeCounts and FaceNodes for additional information.

```
/* The FaceNodeCounts array is used to describe the number of
 * nodes in each face of the zone. The first value in the array
 * is the number of nodes in Face 1, the second value is the
 * number of nodes in Face 2 and so forth. In this example, each
 * face of the zone has four nodes.
 */

INTEGER4 *FaceNodeCounts_Rect = new INTEGER4[NumFaces_Rect];
//For this particular zone, each face has the 4 nodes
for (INTEGER4 ii = 0; ii < NumFaces_Rect; ii++)
   FaceNodeCounts_Rect[ii] = 4;

/* The FaceNodes array is used to specify the nodes that compose
 * each face.  For each face (n of N), the number of nodes used
 * to define the face is specified by the nth value in the
 * FaceNodeCounts array. For example, if the first value in the
 * FaceNodeCounts array is 4 (indicating Face 1 is composed of
 * four nodes), the first four values in the FaceNodes array are
 * the node numbers of the nodes in Face 1.
 *
 * ------------
```

```
 * WARNING
 * When providing the node numbers for each face, you must
 * provide the node numbers in a consistent order (either
 * clockwise or counter-clockwise. Providing the node numbers
 * out of order results in contorted faces.
 * ------------
 */

INTEGER4 *FaceNodes_Rect  = new INTEGER4[TotalNumFaceNodes_Rect];

//Nodes for Face 1
FaceNodes_Rect[0]  = 1;
FaceNodes_Rect[1]  = 2;
FaceNodes_Rect[2]  = 3;
FaceNodes_Rect[3]  = 4;

//Nodes for Face 2
FaceNodes_Rect[4]  = 1;
FaceNodes_Rect[5]  = 4;
FaceNodes_Rect[6]  = 8;
FaceNodes_Rect[7]  = 5;

//Nodes for Face 3
FaceNodes_Rect[8]  = 5;
FaceNodes_Rect[9]  = 8;
FaceNodes_Rect[10]  = 7;
FaceNodes_Rect[11]  = 6;

//Nodes for Face 4
FaceNodes_Rect[12]  = 2;
FaceNodes_Rect[13]  = 6;
FaceNodes_Rect[14]  = 7;
FaceNodes_Rect[15]  = 3;

//Nodes for Face 5
FaceNodes_Rect[16]  = 6;
FaceNodes_Rect[17]  = 2;
FaceNodes_Rect[18]  = 1;
FaceNodes_Rect[19]  = 5;

//Nodes for Face 6
FaceNodes_Rect[20]  = 3;
FaceNodes_Rect[21]  = 7;
FaceNodes_Rect[22]  = 8;
FaceNodes_Rect[23]  = 4;
```

When providing the node numbers for each face, you must provide the node numbers

> in a consistent order (either clockwise or counter-clockwise. Providing the node numbers out of order results in contorted faces.

Step 5 Specify the neighboring elements for Zone 1

The next step for writing out the polyhedral data is to define the right and left neighboring elements for each face. The neighboring elements can be determined using the right-hand rule. For each face, place your right-hand along the face with your fingers pointing the direction of incrementing node numbers (i.e. from Node 1 to Node 2). The right side of your hand will indicate the right element, and the left side of your hand will indicate the left element. Refer to FaceRightElems and FaceLeftElems for details.

The number zero is used to indicate that there isn't an element on that side of the face. A negative number is used when the neighboring element is in another zone. The value of the negative number along with the FaceBndryConnectionCounts array points to the position in the FaceBoundaryConnectionElems and FaceBoundaryConnectionZones arrays that defines the element and zone numbers of the neighboring element. Refer to Step 6 for details.

Because of the way we numbered the nodes and faces, the right element for every face (except the face connected to the arrowhead) is the element itself (Element 1) and the left element is "no-neighboring element" (Element 0).

```
INTEGER4 *FaceLeftElems_Rect  = new INTEGER4[NumFaces_Rect];
INTEGER4 *FaceRightElems_Rect = new INTEGER4[NumFaces_Rect];

/* Since this zone has just one element, all leftelems are
 * NoNeighboring Element and all right elems are itself
 */
for (INTEGER4 ii = 0; ii < NumFaces_Rect; ii++)
{
    FaceRightElems_Rect[ii] = 1;
    FaceLeftElems_Rect[ii]  = 0;
}

/* The negative value in the FaceLeftElems array indicates that
 * the face is connected to an element in another zone. In this
 * case, Face 4 is connected to a face in Zone 2 (to be defined
 * later in the example). The FaceBoundaryConnectionElems array
 * lists all of the element numbers in other zones that the
 * current zone shares boundary connections with. Similarly, the
 * FaceBoundaryConnectionZones array lists all of the zone numbers
 * with which the current zone shares boundaries.  A negative
 * value in the FaceLeftElems or FaceRightElems array indicates
 * the position within these arrays that defines the neighboring
 * element and zone for a face.
 *
```

```
 * For example, if the FaceBoundaryConnectionElems array is:
 * [1 8 2] and the FaceBoundaryConnectionZones array is: [2 5 3],
 * a FaceLeftElems or FaceRightElems value of -2 indicates that
 * the face in question has a boundary connection with Element 8
 * in Zone 5.
 */
FaceLeftElems_Rect[3]  = -1;

INTEGER4 NumFaces_Rect32 = (INTEGER4)NumFaces_Rect;
I = TECPOLYFACE142(&NumFaces_Rect32,
                   FaceNodeCounts_Rect,
                   FaceNodes_Rect,
                   FaceLeftElems_Rect,
                   FaceRightElems_Rect);
```

After defining the face nodes, we call TECPOLYFACE to write it to the file.

Step 6 Define boundary connections for Zone 1

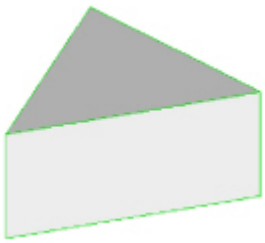
The last step for defining the rectangular solid is to describe the boundary connections and call TECPOLYBCONN.

```
/* The FaceBndryConnectionCounts array is used to define the
 * number of boundary connections for each face that has a
 * boundary connection. For example, if a zone has three boundary
 * connections in total (NumConnectedBoundaryFaces), two of those
 * boundary connections are in one face, and the remaining
 * boundary connection is in a second face, the
 * FaceBndryConnectionCounts array would be: [2 1].
 * In this example, the total number of connected boundary faces
 * (specified via TECPOLYZNE) is equal to one, so the
 * FaceBoundaryConnectionCounts array contains a single value (1).
 */
INTEGER4 *FaceBndryConnCounts_Rect = new
INTEGER4[NumConnBndryFaces_Rect];
FaceBndryConnCounts_Rect[0] = 1;

/* The value(s) in the FaceBndryConnectionElems and
* FaceBndryConnectionZones arrays specify the element number and
* zone number, respectively, that a given boundary connection is
* connected to. In this case, the boundary connection face is
* connected to Element 1 in Zone 2.
*/
INTEGER4 *FaceBndryConnElems_Rect = new INTEGER4[TotalNumBndryConns_Rect];
INTEGER4 *FaceBndryConnZones_Rect = new INTEGER4[TotalNumBndryConns_Rect];
```

```
FaceBndryConnElems_Rect[0]  = 1;
FaceBndryConnZones_Rect[0]  = 2;

I = TECPOLYBCONN142(&NumConnBndryFaces_Rect,
                    FaceBndryConnCounts_Rect,
                    FaceBndryConnElems_Rect,
                    FaceBndryConnZones_Rect);

/* cleanup */
delete X_Rect;
delete Y_Rect;
delete Z_Rect;
delete P_Rect;
delete FaceNodeCounts_Rect;
delete FaceNodes_Rect;
delete FaceLeftElems_Rect;
delete FaceRightElems_Rect;
delete FaceBndryConnCounts_Rect;
delete FaceBndryConnElems_Rect;
delete FaceBndryConnZones_Rect;
```

Step 7 Create Zone 2

The data for Zone 1 has been written to the data file, so we are ready to create Zone 2. For simplicity, we will reuse many of the variables from creating multi-poly data that are not relevant to this tutorial.



Zone 2 (the arrowhead or prism) has a single element composed of six nodes and five faces.

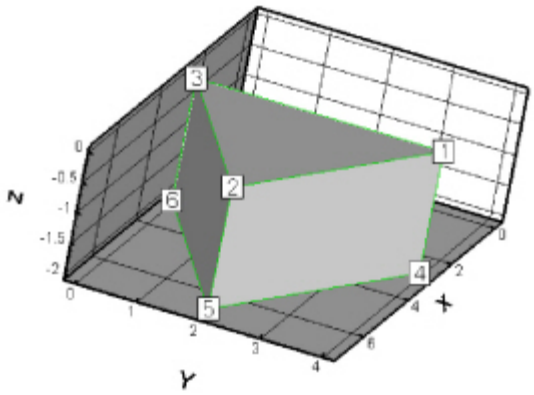
```
//TECPOLYZNE Parameters
INTEGER4 NumPts_Prism    = 6;
INTEGER4 NumElems_Prism  = 1;
INTEGER8 NumFaces_Prism  = 5;

/* The prism is composed of two triangular faces and three
 * rectangular faces.  The total number of face nodes is the sum
 * of the nodes in each triangular face (2 times 3) and the nodes
 * in each rectangular face (3 times 4).
 */
```

```
INTEGER8 TotalNumFaceNodes_Prism  = 18;

/* As with Zone 1, Zone 2 has one connected boundary face, the
 * face that is connected to Zone 1.
 */
INTEGER4 NumConnBndryFaces_Prism  = 1;

/* In this case, we have set the total number of boundary
 * connections for the connected face to two. The first boundary
 * connection is the connection to Zone 1.  The second boundary
 * connection is used to indicate that the face is only partially
 * obscured by the face from Zone 1. If we omitted the second
 * boundary connection, the connected face of the prism would
 * disappear if the rectangular zone was deactivated.
 */
INTEGER4 TotalNumBndryConns_Prism = 2;

I = TECPOLYZNE142((char*)"Zone 2: Prism",
                  &ZoneType,
                  &NumPts_Prism,
                  &NumElems_Prism,
                  &NumFaces_Prism,
                  &TotalNumFaceNodes_Prism,
                  &SolutionTime,
                  &StrandID,
                  &Unused,
                  &NumConnBndryFaces_Prism,
                  &TotalNumBndryConns_Prism,
                  NULL,
                  ValueLocation,
                  NULL,
                  &SharConn);
```

Step 8 Specify the variable values for Zone 2

Now that the zone has been created, we must write the variable values to the file by calling `TECDAT`. While there are more elegant ways to define the grid coordinates for the prism, the values are defined explicitly in order to keep the example relatively simple.

Using the picture below, define the variable values.




```
double *X_Prism = new double[NumPts_Prism];
double *Y_Prism = new double[NumPts_Prism];
double *Z_Prism = new double[NumPts_Prism];

/* Set the X and Y variable values, one z-plane at a time */
double ZVal = 0;
for (INTEGER4 ii = 0; ii < 2; ii++)
{
   // triangle in Z=ZVal plane
    X_Prism[3*ii]  = 3;
    Y_Prism[3*ii]  = 4;
    Z_Prism[3*ii]  = ZVal;

    X_Prism[3*ii+1] = 7;
    Y_Prism[3*ii+1] = 2;
    Z_Prism[3*ii+1] = ZVal;

    X_Prism[3*ii+2] = 3;
    Y_Prism[3*ii+2] = 0;
    Z_Prism[3*ii+2] = ZVal;

    ZVal = ZVal - 2;
}

/* When we called TECPOLYZNE, we specified that the variable 4
 * (pressure) is cell-centered. As such, only NumElements number
 * of values needs to be written to the data file for the pressure
 * variable.
 */
double *P_Prism = new double[NumElems_Prism];
P_Prism[0] = 20;

I = TECDAT142(&NumPts_Prism,  X_Prism, &IsDouble);
I = TECDAT142(&NumPts_Prism,  Y_Prism, &IsDouble);
I = TECDAT142(&NumPts_Prism,  Z_Prism, &IsDouble);
```

```
I = TECDAT142(&NumElems_Prism, P_Prism, &IsDouble);
```

Step 9 Define the face map for the arrowhead

Before creating the data set, we have defined the node numbers, face numbers and element numbers. Using the following figure, specify the nodes that define each face.

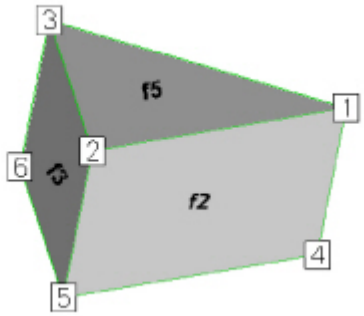


Figure 13. The arrowhead with three faces visible (Face 2, Face 3 and Face 5). The remaining rectangular face is Face 1, and the remaining triangular face is Face 4).

The faces are created from the data file format using the FaceNodeCounts and FaceNodes array. The FaceNodeCounts array specifies the number of nodes contained in each face. The first value in the array is the number of nodes in Face 1, followed by the number of nodes in Face 2, and so forth. The FaceNodes array lists the node numbers in each face. The FaceNodes array first lists all of the nodes in Face 1, followed by all of the nodes in Face 2, and so forth.

In this example, Face 1 is composed of four nodes (Node 1, Node 3, Node 6 and Node 4). As such, the first value in the FaceNodeCounts array is "4" and the first four values in the FaceNodes array are [1, 3, 6, 4].

```
INTEGER4 *FaceNodeCounts_Prism = new INTEGER4[NumFaces_Prism];
INTEGER4 *FaceNodes_Prism      = new INTEGER4[TotalNumFaceNodes_Prism];

/* Because of the way we chose to number our faces, the first
 * three faces are rectangular and the last two are triangular.
 * The numbering of the faces is arbitrary, but the faces must
 * be referred to consistently.
 */
for (INTEGER4 ii = 0; ii < 3; ii++)
    FaceNodeCounts_Prism[ii] = 4;

for (INTEGER4 ii = 3; ii < NumFaces_Prism; ii++)
    FaceNodeCounts_Prism[ii] = 3;

//Nodes for Face 1
FaceNodes_Prism[0]  = 1;
```

```
FaceNodes_Prism[1]  = 3;
FaceNodes_Prism[2]  = 6;
FaceNodes_Prism[3]  = 4;

//Nodes for Face 2
FaceNodes_Prism[4]  = 1;
FaceNodes_Prism[5]  = 4;
FaceNodes_Prism[6]  = 5;
FaceNodes_Prism[7]  = 2;

//Nodes for Face 3
FaceNodes_Prism[8]  = 3;
FaceNodes_Prism[9]  = 2;
FaceNodes_Prism[10]  = 5;
FaceNodes_Prism[11]  = 6;

//Nodes for Face 4
FaceNodes_Prism[12]  = 5;
FaceNodes_Prism[13]  = 4;
FaceNodes_Prism[14]  = 6;

//Nodes for Face 5
FaceNodes_Prism[15]  = 1;
FaceNodes_Prism[16]  = 2;
FaceNodes_Prism[17]  = 3;
```

Step 10 Specify the neighboring elements for Zone 2

Now that we have defined the nodes that compose each face, we must specify the element on either side of each face. The neighboring elements can be determined using the right-hand rule. For each face, place your right-hand along the face with your fingers pointing the direction of incrementing node numbers (i.e. from Node 1 to Node 2). The right side of your hand will indicate the right element, and the left side of your hand will indicate the left element. Refer to FaceRightElems and FaceLeftElems for details.

The number zero is used to indicate that there isn't an element on that side of the face. A negative number is used when the neighboring element is in another zone. The value of the negative number points to the position in the FaceBoundaryConnectionElems and FaceBoundaryConnectionZones arrays that defines the element and zone numbers of the neighboring element. Refer to Step 11 for details.

Because of the way we numbered the nodes and faces, the right element for every face (except the face connected to the rectangular solid) is the element itself (Element 1) and the left element is "no-neighboring element" (Element 0).

```
/* Since this zone has just one element, all leftelems are
```

```
 * NoNeighboring Element and all right elems are itself.
 */
INTEGER4 *FaceLeftElems_Prism  =  new INTEGER4[NumFaces_Prism];
INTEGER4 *FaceRightElems_Prism =  new INTEGER4[NumFaces_Prism];

for (INTEGER4 ii = 0; ii < NumFaces_Prism; ii++)
{
 FaceRightElems_Prism[ii] = 1;
 FaceLeftElems_Prism[ii]  = 0;
}

/* The negative value in the FaceLeftElems array indicates that
* the face is connected to an element in another zone. In this
* case, Face 1 is connected to a face in Zone 1 (as indicated in
* Line 6). The FaceBoundaryConnectionElems array lists all of
* the element numbers in other zones that the current zone shares
* boundary connections with. Similarly, the
* FaceBoundaryConnectionZones array lists all of the zone numbers
* with which the current zone shares boundaries.  A negative
* value in the FaceLeftElems or FaceRightElems array indicates
* the position within these arrays that defines the neighboring
* element and zone for a face.
*/
FaceLeftElems_Prism[0]  = -1;

INTEGER4 NumFaces_Prism32 = (INTEGER4)NumFaces_Prism;
I = TECPOLYFACE142(&NumFaces_Prism32,
                   FaceNodeCounts_Prism,
                   FaceNodes_Prism,
                   FaceLeftElems_Prism,
                   FaceRightElems_Prism);
```

Step 11 boundary connections for Zone 2

The last step for creating Zone 2 is to specify the boundary connections.

```
INTEGER4 *FaceBndryConnCounts_Prism = new INTEGER4[NumConnBndryFaces_Prism];
FaceBndryConnCounts_Prism[0] = 2;

INTEGER4 *FaceBndryConnElems_Prism = new INTEGER4[TotalNumBndryConns_Prism];
INTEGER4 *FaceBndryConnZones_Prism = new INTEGER4[TotalNumBndryConns_Prism];

/* As previously mentioned, a connected boundary face is a face
 * that has either multiple neighboring faces or neighbor(s) that
 * belong to another zone.  Those cases are sufficient when the
 * combination of all of the face neighbors completely cover the
```

```
 * face. However, there are some cases (such as the bottom of the
 * arrowhead) where the face is not completely covered by its
 * neighbors. In those cases the face is referred to as partially
 * obscured. A partially obscured face is indicated by
 * incrementing the value in TotalNumConnectedBoundaryFaces and
 * entering a value of 0 in both the FaceBndryConnectionElems and
 * FaceBoundaryConnectionZones arrays for the boundary connection
 * for the partially obscured face.
 */
FaceBndryConnElems_Prism[0]  = 0;
FaceBndryConnZones_Prism[0]  = 0;

/* Indicates that Face 1 is connected to Element 1 in Zone 1. */
FaceBndryConnElems_Prism[1]  = 1;
FaceBndryConnZones_Prism[1]  = 1;

I = TECPOLYBCONN142(&NumConnBndryFaces_Prism,
                    FaceBndryConnCounts_Prism,
                    FaceBndryConnElems_Prism,
                    FaceBndryConnZones_Prism);

/* cleanup */
delete X_Prism;
delete Y_Prism;
delete Z_Prism;
delete P_Prism;
delete FaceNodeCounts_Prism;
delete FaceNodes_Prism;
delete FaceLeftElems_Prism;
delete FaceRightElems_Prism;
delete FaceBndryConnCounts_Prism;
delete FaceBndryConnElems_Prism;
delete FaceBndryConnZones_Prism;
```

Step 12 Close the file

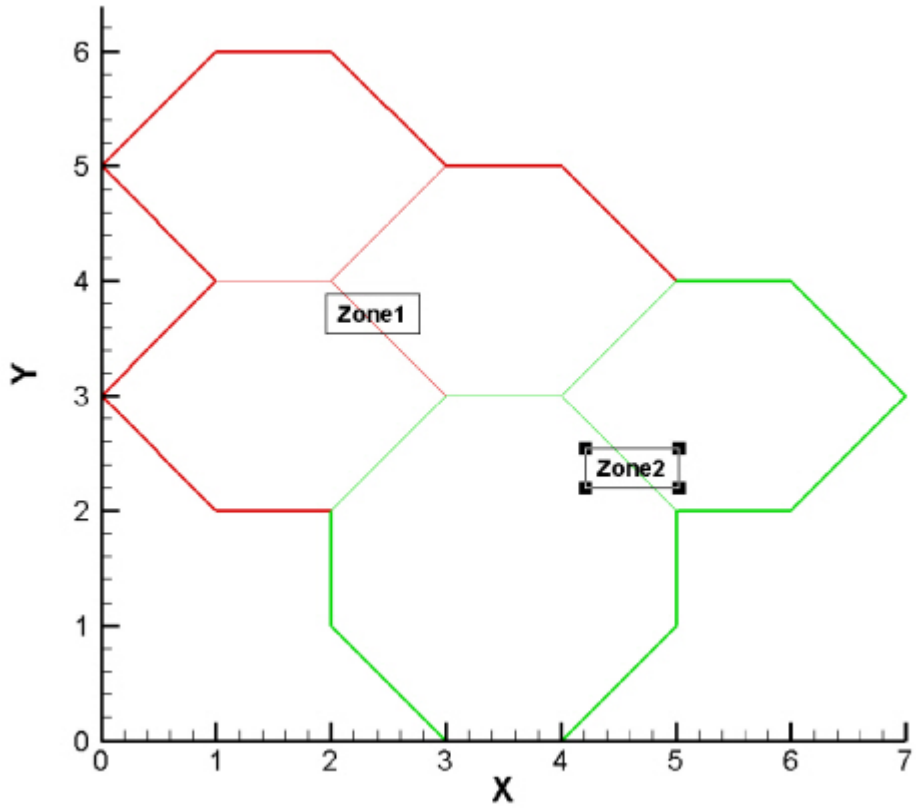
Call TECEND to close the file.

```
I = TECEND142();
```

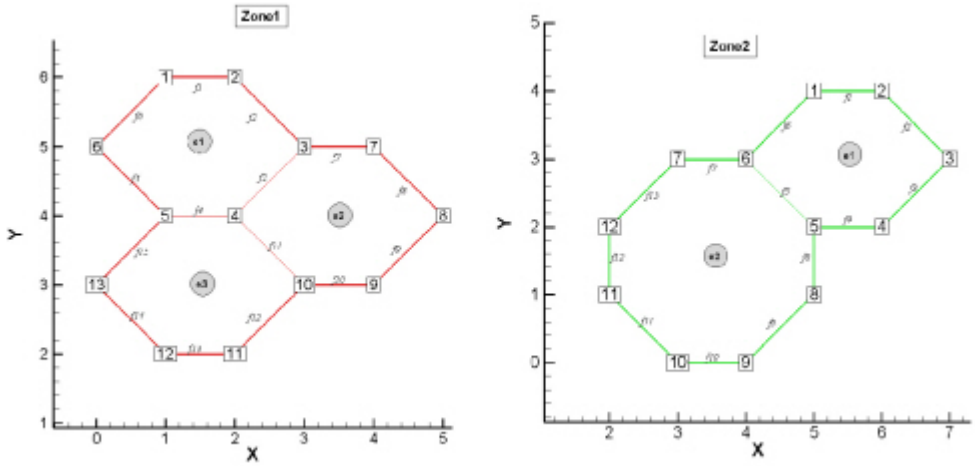
Multiple Polygonal Zones

The following example demonstrates how to create multiple polygonal zones. The example covers: creating a zone where each element contains a different number of nodes, boundary connections and varying variable locations (cell-centered versus nodal).

The code in this example produces the following plot:



Before beginning to create a polyhedral data file, you should assign a number to each node, face, element and zone. The numbering system is used to determine the order that the information is supplied to Tecplot. You may assign any order you would like. However, once you have supplied information to Tecplot, you cannot change the number configuration. For this example, we have selected the numbering system shown below:



Zone 1 has a total of three elements, thirteen unique nodes and fifteen faces. Zone 2 has two elements, twelve nodes and thirteen faces.

In order to keep the example as simple as possible, error checking is not included. If you plan to compile this example, be sure to include: TECIO.h.

For complete details on the parameters used and the function syntax for each TecIO function, refer to Binary Data File Function Reference. When creating a binary data file using the TecIO library, the functions must be called in a specific order. Refer to Binary Data File Function Calling Sequence for details.

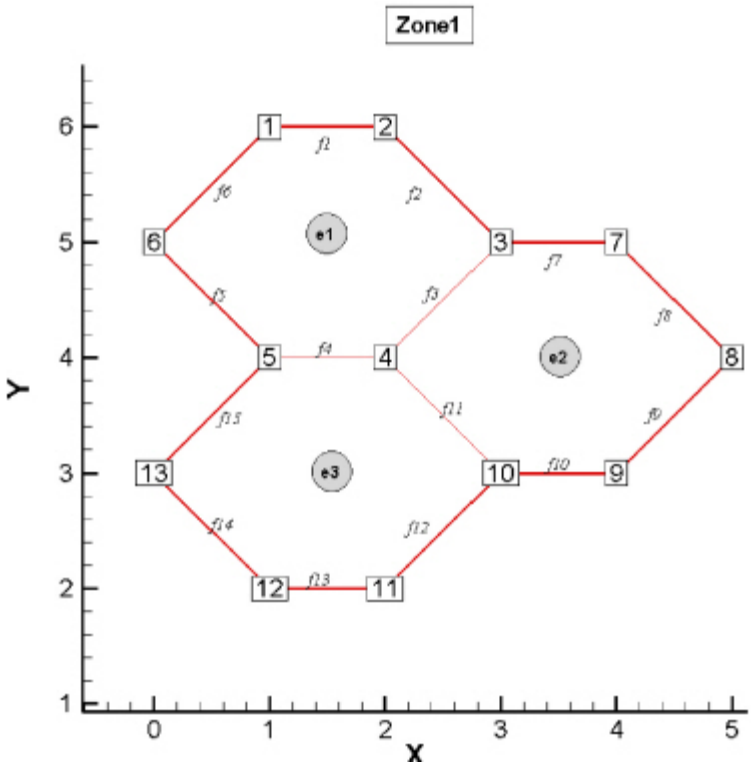
Step 1 Initialize the Data File

The first step for creating a binary data file using the TecIO library is to initialize and open the data file by calling TECINI

```
INTEGER4 I; /* use to check return values */

INTEGER4 Debug      = 1;
INTEGER4 VIsDouble  = 0;
INTEGER4 FileType   = 0;
INTEGER4 FileFormat = 0; // 0 == PLT, 1 == SZPLT

I = TECINI142((char*)"Example: Multiple polygonal zones",
              (char*)"X Y P",  /* Defines the variables for the data file.
                                * Each zone must contain each of the vars
                                * listed here. The order of the variables
                                * in the list is used to define the
                                * variable number (e.g. X is Variable).
                                * When referring to variables in other
                                * TecIO functions, you will refer to the
                                * variable by its number.
                                */
              (char*)"multiplepolygons-HexAndOct.plt",
              (char*)".",  /* scratch directory */
              &FileFormat,
              &FileType,
              &Debug,
              &VIsDouble);
```

Step 2 Create Zone 1 (3 Hexagons)

The first step toward creating Zone 1 is to call TECPOLYZNE. TECPOLYZNE is used to initialize the zone and specify parameters that apply to the entire zone (e.g. number of nodes, number of elements and variable location).




```
/* TECPOLYZNE Parameters */
INTEGER4 ZoneType             = 6;   /* FE Polygon */
INTEGER4 NumPts_Z1            = 13;  /* the number of unique
                                        * nodes in the zone.
                                        */
INTEGER4 NumElems_Z1  = 3;
INTEGER8 NumFaces_Z1          = 15;  /* the number of unique
                                        * faces in the zone.
                                        */
double  SolutionTime          = 0.0;
INTEGER4 StrandID             = 0;
INTEGER4 unused               = 0;  // ParentZone is no longer used
INTEGER4 SharConn             = 0;

INTEGER4 ValueLocation[3]     = { 1, 1, 0 };

/* For a polygonal zone, the total number of face nodes is
 * twice the total number of faces. This is because each face
 * is composed of exactly two nodes.
 */
INTEGER8 TotalNumFaceNodes_Z1  = 2 * NumFaces_Z1;

/* A boundary face is a face that is neighbored by an element
 * or elements in another zone or zone(s). In Zone 1, Face 9,
 * Face 10 and Face 12 have a neighbor in Zone 2. Therefore,
 * the total number of boundary faces is "3".
```

```
 */
INTEGER4 TotalNumBndryFaces_Z1 = 3;

/* Each boundary face has one or more boundary connections. A
 * boundary connection is defined as another element in another
 * zone. Face 9 has a boundary connection with Element 1 in
 * Zone 2. In this example, each boundary face is connected to
 * one other element, so the total number of boundary
 * connections is equivalent to the total number of boundary
 * faces (3).
 */
INTEGER4 TotalNumBndryConns_Z1 = 3;

I = TECPOLYZNE142((char*)"Zone 1: 3 Hexagons", /* Specifies the name of
                                                * the entire dataset.
                                                * When the file is loaded
                                                * into Tecplot, the value is
                                                * available via the Data
                                                * Set Info dialog.
                                                */
                  &ZoneType,
                  &NumPts_Z1,
                  &NumElems_Z1,
                  &NumFaces_Z1,
                  &TotalNumFaceNodes_Z1,
                  &SolutionTime,
                  &StrandID,
                  &unused,
                  &TotalNumBndryFaces_Z1,
                  &TotalNumBndryConns_Z1,
                  NULL,
                  ValueLocation,
                  NULL,
                  &SharConn);
```

Step 3 Specify the variable values for Zone 1

The variable values are written to the data file via the TECDAT function. For each variable you must provide either a total number of values equivalent to NumPts (if the variables are nodal) or a total number of values equivalent to NumElements (if the variables are cell-centered). The variable location is specified by the VarLocation parameter in TECPOLYZNE. In this example, X and Y are nodal variables and P is cell-centered.

The order in which the variable values must be provided is established by the numbering scheme (specified at the beginning of the example). The first value for each nodal variable (X and Y) corresponds to Node 1, the second value corresponds to Node 2 and so forth. The first value for the cell-centered value is for Element 1, the second value is for the second element or cell and so forth.

In order for the example to be easily followed, the grid coordinates are explicitly defined. When working with larger data sets, you will likely wish to use equations to define your coordinates. Refer to the picture in Step 2 for the X and Y coordinate values for Zone 1.

```
/* TECDAT Parameters */
double *X_Z1 = new double[NumPts_Z1];
double *Y_Z1 = new double[NumPts_Z1];

X_Z1[0]  = 1;
Y_Z1[0]  = 6;

X_Z1[1]  = 2;
Y_Z1[1]  = 6;

X_Z1[2]  = 3;
Y_Z1[2]  = 5;

X_Z1[3]  = 2;
Y_Z1[3]  = 4;

X_Z1[4]  = 1;
Y_Z1[4]  = 4;

X_Z1[5]  = 0;
Y_Z1[5]  = 5;

X_Z1[6]  = 4;
Y_Z1[6]  = 5;

X_Z1[7]  = 5;
Y_Z1[7]  = 4;

X_Z1[8]  = 4;
Y_Z1[8]  = 3;

X_Z1[9]  = 3;
Y_Z1[9]  = 3;

X_Z1[10] = 2;
Y_Z1[10] = 2;

X_Z1[11] = 1;
Y_Z1[11] = 2;

X_Z1[12] = 0;
Y_Z1[12] = 3;
```

```
double *P_Z1 = new double[NumElems_Z1];
P_Z1[0] = 2;
P_Z1[1] = 4;
P_Z1[2] = 5;

INTEGER4  IsDouble = 1;
I = TECDAT142(&NumPts_Z1,  X_Z1, &IsDouble);
I = TECDAT142(&NumPts_Z1,  Y_Z1, &IsDouble);
I = TECDAT142(&NumElems_Z1, P_Z1, &IsDouble);
delete X_Z1;
delete Y_Z1;
delete P_Z1;
```

Step 4 Specify the face map data for Zone 1

Use the picture in Step 2 to specify the nodes that compose each face. The first two values in the face node array define Face 1, the next two define Face 2, and so on.

```
/* TecPolyFace Parameters */

/* Create a FaceNodes array, dimensioned by the total number
 * of face nodes in the zone.
 */
INTEGER4 *FaceNodes_Z1 = new INTEGER4[TotalNumFaceNodes_Z1];

/* Face Nodes for Element 1 */
FaceNodes_Z1[0]  = 1;
FaceNodes_Z1[1]  = 2;

FaceNodes_Z1[2]  = 2;
FaceNodes_Z1[3]  = 3;

FaceNodes_Z1[4]  = 3;
FaceNodes_Z1[5]  = 4;

FaceNodes_Z1[6]  = 4;
FaceNodes_Z1[7]  = 5;

FaceNodes_Z1[8]  = 5;
FaceNodes_Z1[9]  = 6;

FaceNodes_Z1[10] = 6;
FaceNodes_Z1[11] = 1;

/* Face Nodes for Element 2 */
FaceNodes_Z1[12] = 3;
```

```
FaceNodes_Z1[13] = 7;

FaceNodes_Z1[14] = 7;
FaceNodes_Z1[15] = 8;

FaceNodes_Z1[16] = 8;
FaceNodes_Z1[17] = 9;

FaceNodes_Z1[18] = 9;
FaceNodes_Z1[19] = 10;

FaceNodes_Z1[20] = 10;
FaceNodes_Z1[21] = 4;

/* Face Nodes for Element 3 */
FaceNodes_Z1[22] = 10;
FaceNodes_Z1[23] = 11;

FaceNodes_Z1[24] = 11;
FaceNodes_Z1[25] = 12;

FaceNodes_Z1[26] = 12;
FaceNodes_Z1[27] = 13;

FaceNodes_Z1[28] = 13;
FaceNodes_Z1[29] = 5;
```

Step 5 Specify the neighboring elements for Zone 1

Now that we have defined the nodes that compose each face, we must specify the element on either side of each face. The neighboring elements can be determined using the right-hand rule. For each face, place your right-hand along the face with your fingers pointing the direction of incrementing node numbers (i.e. from Node 1 to Node 2). The right side of your hand will indicate the right element, and the left side of your hand will indicate the left element. Refer to FaceRightElems and FaceLeftElems for details.

The number zero is used to indicate that there isn't an element on that side of the face. A negative number is used when the neighboring element is in another zone. The value of the negative number points to the position in the FaceBoundaryConnectionElems and FaceBoundaryConnectionZones arrays that defines the element and zone numbers of the neighboring element. Refer to Step 6 for details.

Because of the way we numbered the nodes and faces, the right element for every face is the element itself. The left element will either be: another element in the zone, "no neighboring element", or an element in Zone 2. The term "no neighboring element" is used to describe a face that is on the edge of the entire data set (not just the zone).

```
INTEGER4 *FaceLeftElems_Z1  = new INTEGER4[NumFaces_Z1];
INTEGER4 *FaceRightElems_Z1 = new INTEGER4[NumFaces_Z1];

/* Left Face Elems for Element 1 */
FaceLeftElems_Z1[0]  = 0;
FaceLeftElems_Z1[1]  = 0;
FaceLeftElems_Z1[2]  = 2;
FaceLeftElems_Z1[3]  = 3;
FaceLeftElems_Z1[4]  = 0;

/* Left Face Elems for Element 2 */
FaceLeftElems_Z1[5]  =  0;
FaceLeftElems_Z1[6]  =  0;
FaceLeftElems_Z1[7]  =  0;
FaceLeftElems_Z1[8]  = -1;
FaceLeftElems_Z1[9]  = -2;
FaceLeftElems_Z1[10] =  3;

/* Left Face Elems for Element 3 */
FaceLeftElems_Z1[11]  = -3;
FaceLeftElems_Z1[12]  =  0;
FaceLeftElems_Z1[13]  =  0;
FaceLeftElems_Z1[14]  =  0;

/* Set Right Face Elems.  Because of the way we numbered the
 * nodes and faces, the right element for every face is the
 * element itself.
 */
for (INTEGER4 ii = 0; ii < 6; ii++)
    FaceRightElems_Z1[ii] = 1;

for (INTEGER4 ii = 6; ii < 11; ii++)
    FaceRightElems_Z1[ii] = 2;

for (INTEGER4 ii = 11; ii <= 14; ii++)
    FaceRightElems_Z1[ii] = 3;

INTEGER4 NumFaces_Z132 = (INTEGER4)NumFaces_Z1;
I = TECPOLYFACE142(&NumFaces_Z132,
                   NULL,  /* Not used for polygon zones */
                   FaceNodes_Z1,
                   FaceLeftElems_Z1,
                   FaceRightElems_Z1);

delete FaceNodes_Z1;
delete FaceLeftElems_Z1;
```

```
delete FaceRightElems_Z1;
```

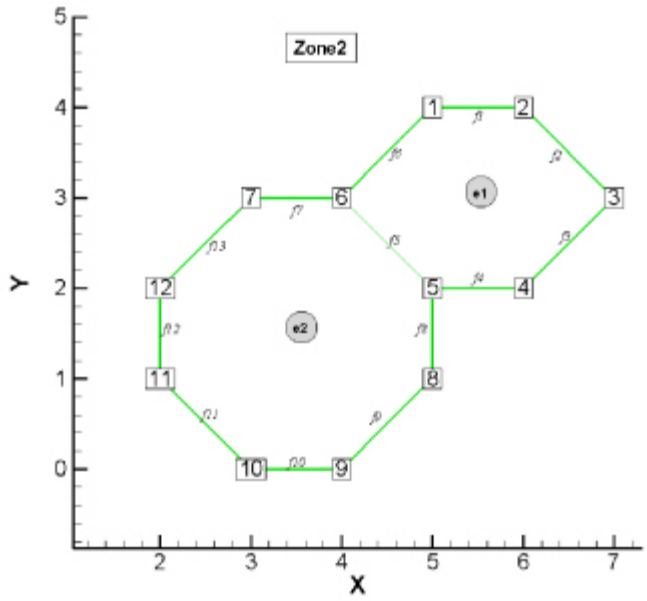
Step 6 Specify the boundary connections for Zone 1

The final step for creating Zone 1 is to define the boundary connections.

```
/* TecPolyBConn Parameters */

/* The FaceBndryConnectionCounts array is used to define the
 * number of boundary connections for each face that has a
 * boundary connection. For example, if a zone has three
 * boundary connections in total (NumConnectedBoundaryFaces),
 * two of those boundary connections are in one face, and the
 * remaining boundary connection is in a second face, the
 * FaceBndryConnectionCounts array would be: [2 1].
 *
 * In this example, the total number of connected boundary
 * faces (specified via TECPOLYZNE) is equal to three. Each
 * boundary face is connected to only one other element,
 * so the FaceBoundaryConnectionCounts array is (1, 1, 1).
 */
INTEGER4 FaceBndryConnectionCounts_Z1[3]  = {1, 1, 1};

/* The value(s) in the FaceBndryConnectionElems and
 * FaceBndryConnectionZones arrays specifies the element number
 * and zone number, respectively, that a given boundary
 * connection is connected to. In this case, the first
 * boundary connection face is connected to Element 1 in Zone 2
 * and the remaining connection is to Element 2 in Zone 2.
 */
INTEGER4 FaceBndryConnectionElems_Z1[3]  = {1, 2, 2};
INTEGER4 FaceBndryConnectionZones_Z1[3]  = {2, 2, 2};

I = TECPOLYBCONN142(&TotalNumBndryFaces_Z1,
                    FaceBndryConnectionCounts_Z1,
                    FaceBndryConnectionElems_Z1,
                    FaceBndryConnectionZones_Z1);
```

Step 7 Create Zone 2

Now that Zone 1 is complete, we are ready to begin creating Zone 2 by calling `TECPOLYZNE`. For simplicity, we are reusing many of the variables that were defined for Zone 1.




```
INTEGER4 NumPts_Z2              = 12; /* number of unique
                                      * nodes in the zone
                                      */
INTEGER4 NumElems_Z2            = 2;
INTEGER8 NumFaces_Z2            = 13; /* number of unique
                                      * faces in the zone
                                      */
INTEGER4 NumFaceConnections_Z2  = 0;

/* In polygonal zones, each face has exactly two nodes */
INTEGER8 TotalNumFaceNodes_Z2   = NumFaces_Z2 * 2;

/* A boundary face is a face that is neighbored by an element or
 * elements from another zone or zone(s). In Zone 2, Face 6,
 * Face 7 and Face 13 have a neighbor in Zone 1. Therefore, the
 * total number of boundary faces is "3".
 */
INTEGER4 TotalNumBndryFaces_Z2  = 3;

/* Each boundary face has one or more boundary connections. In
 * this example, each boundary face is connected to one other
 * element (i.e. the number of boundary faces and the number of
 * boundary connections is one-to-one).
 */
INTEGER4 TotalNumBndryConns_Z2  = 3;

I = TECPOLYZNE142((char*)"Zone 2: 1 Hexagon and 1 Octagon",
                  &ZoneType,
                  &NumPts_Z2,
                  &NumElems_Z2,
```

```
                &NumFaces_Z2,
                &TotalNumFaceNodes_Z2,
                &SolutionTime,
                &StrandID,
                &unused,
                &TotalNumBndryFaces_Z2,
                &TotalNumBndryConns_Z2,
                NULL,
                ValueLocation,
                NULL,
                &SharConn);
```

Step 8 Specify the variable values for Zone 2

The variable values are written to the data file via the TECDAT function. For each variable you must provide either a total number of values equivalent to NumPts (if the variables are nodal) or equivalent to NumElements (if the variables are cell-centered). The variable location is specified by the VarLocation parameter in TECPOLYZNE. In this example, X and Y are nodal variables and P is cell-centered.

The order in which the variable values must be provided is established by the numbering scheme specified at the beginning of the example. The first value for each nodal variable (X and Y) corresponds to Node 1, the second value corresponds to Node 2 and so forth. The first value for the cell-centered value is for Element 1, the second value is for the second element or cell and so forth.

In order for the example to be easily followed, the grid coordinates are explicitly defined. When working with larger data sets, you will likely wish to use equations to define your coordinates. Refer to the picture in Step 7 for the X and Y coordinate values for Zone 2.

```
double *X_Z2 = new double[NumPts_Z2];
double *Y_Z2 = new double[NumPts_Z2];

X_Z2[0]  = 5;
Y_Z2[0]  = 4;

X_Z2[1]  = 6;
Y_Z2[1]  = 4;

X_Z2[2]  = 7;
Y_Z2[2]  = 3;

X_Z2[3]  = 6;
Y_Z2[3]  = 2;

X_Z2[4]  = 5;
Y_Z2[4]  = 2;
```

```
X_Z2[5]  = 4;
Y_Z2[5]  = 3;

X_Z2[6]  = 3;
Y_Z2[6]  = 3;

X_Z2[7]  = 5;
Y_Z2[7]  = 1;

X_Z2[8]  = 4;
Y_Z2[8]  = 0;

X_Z2[9]  = 3;
Y_Z2[9]  = 0;

X_Z2[10]  = 2;
Y_Z2[10]  = 1;

X_Z2[11]  = 2;
Y_Z2[11]  = 2;

/* In the call to TECPOLYZNE, P was set to a cell centered variable.
 * As such, only two values need to be defined.
 */
double  *P_Z2 = new double[NumPts_Z2];

P_Z2[0] = 8;
P_Z2[1] = 6;

I = TECDAT142(&NumPts_Z2,  X_Z2, &IsDouble);
I = TECDAT142(&NumPts_Z2,  Y_Z2, &IsDouble);
I = TECDAT142(&NumElems_Z2, P_Z2, &IsDouble);

delete X_Z2;
delete Y_Z2;
delete P_Z2;
```

Step 9 Specify the face map for Zone 2

Use the picture in Step 7 to specify which nodes compose which face. The first two values in the face node array define Face 1, the next two define Face 2, and so on.

```
INTEGER4 *FaceNodes_Z2;
FaceNodes_Z2 = new INTEGER4[TotalNumFaceNodes_Z2];
```

```
/* Face Nodes for Element 1 */
FaceNodes_Z2[0]  = 1;
FaceNodes_Z2[1]  = 2;

FaceNodes_Z2[2]  = 2;
FaceNodes_Z2[3]  = 3;

FaceNodes_Z2[4]  = 3;
FaceNodes_Z2[5]  = 4;

FaceNodes_Z2[6]  = 4;
FaceNodes_Z2[7]  = 5;

FaceNodes_Z2[8]  = 5;
FaceNodes_Z2[9]  = 6;

FaceNodes_Z2[10] = 6;
FaceNodes_Z2[11] = 1;

/* Face Nodes for Element 2 */
FaceNodes_Z2[12] = 7;
FaceNodes_Z2[13] = 6;

FaceNodes_Z2[14] = 5;
FaceNodes_Z2[15] = 8;

FaceNodes_Z2[16] = 8;
FaceNodes_Z2[17] = 9;

FaceNodes_Z2[18] = 9;
FaceNodes_Z2[19] = 10;

FaceNodes_Z2[20] = 10;
FaceNodes_Z2[21] = 11;

FaceNodes_Z2[22] = 11;
FaceNodes_Z2[23] = 12;

FaceNodes_Z2[24] = 12;
FaceNodes_Z2[25] = 7;
```

Step 10 Specify the neighboring elements for Zone 2

Now that we have defined the nodes that compose each face, we must specify the element on either side of each face. The neighboring elements can be determined using the right-hand rule. For each face, place your right-hand along the face with your fingers pointing the direction of incrementing node numbers (i.e. from Node 1 to Node 2). The right side of your hand will indicate the right element,

and the left side of your hand will indicate the left element. Refer to FaceRightElems and FaceLeftElems for details.

The number zero is used to indicate that there isn't an element on that side of the face. A negative number is used when the neighboring element is in another zone. The value of the negative number points to the position in the FaceBoundaryConnectionElems and FaceBoundaryConnectionZones arrays that defines the element and zone numbers of the neighboring element. Refer to Step 11 for details.

Because of the way we numbered the nodes and faces, the right element for every face is the element itself. The left element will either be: another element in the zone, "no neighboring element", or an element in Zone 2. The term "no neighboring element" is used to describe a face that is on the edge of the entire data set (not just the zone).

```
/* Specify the right and left neighboring elements.
 * The neighboring elements can be determined using the
 * right-hand rule. For each face, place your right-hand along
 * the face with your fingers pointing the direction of
 * incrementing node numbers (i.e. from Node 1 to Node 2). The
 * right side of your hand will indicate the right element,
 * and the left side of your hand will indicate the left
 * element. A value of zero indicates that there is no
 * neighboring element on that side.  A negative value
 * indicates that the neighboring element is in another zone.
 * The number is a pointer into the FaceBndryConnectionElems
 * and FaceBndryConnectionZones arrays.
 */

INTEGER4 *FaceLeftElems_Z2  = new INTEGER4[NumFaces_Z2];
INTEGER4 *FaceRightElems_Z2 = new INTEGER4[NumFaces_Z2];

/* Left Face Elems for Element 1 */
FaceLeftElems_Z2[0]  =  0;
FaceLeftElems_Z2[1]  =  0;
FaceLeftElems_Z2[2]  =  0;
FaceLeftElems_Z2[3]  =  0;
FaceLeftElems_Z2[4]  =  2;
FaceLeftElems_Z2[5]  = -1;

/* Left Face Elems for Element 2 */
FaceLeftElems_Z2[6]  = -2;
FaceLeftElems_Z2[7]  =  0;
FaceLeftElems_Z2[8]  =  0;
FaceLeftElems_Z2[9]  =  0;
FaceLeftElems_Z2[10] =  0;
FaceLeftElems_Z2[11] =  0;
FaceLeftElems_Z2[12] = -3;
```

```
/* Set Right Face Elems.  Because of the way we numbered the
 * nodes and faces, the right element for every face is the
 * element itself. */
for (INTEGER4 ii = 0; ii < 6; ii++)
    FaceRightElems_Z2[ii]  = 1;

for (INTEGER4 ii = 6; ii < 13; ii++)
    FaceRightElems_Z2[ii]  = 2;

INTEGER4 NumFaces_Z232 = (INTEGER4)NumFaces_Z2;
I = TECPOLYFACE142(&NumFaces_Z232,
                   NULL,
                   FaceNodes_Z2,
                   FaceLeftElems_Z2,
                   FaceRightElems_Z2);

delete FaceNodes_Z2;
delete FaceLeftElems_Z2;
delete FaceRightElems_Z2;
```

Step 11 Specify the Boundary Connections for Zone 2

The final step for creating Zone 2 is to define the boundary connections

```
/* The FaceBndryConnectionCounts array is used to define the
 * number of boundary connections for each face that has a
 * boundary connection. In this example, the total number of
 * connected boundary faces (specified via TECPOLYZNE) is equal to
 * three. Each boundary face is connected to only one other
 * element, so the FaceBoundaryConnectionCounts array is
 * (1, 1, 1).
 */
INTEGER4 FaceBndryConnectionCounts_Z2[3]  = {1, 1, 1};

/* The value(s) in the FaceBndryConnectionElems and
 * FaceBndryConnectionZones arrays specifies that element
 * number and zone number, respectively, that a given boundary
 * connection is connected to. In this case, the first boundary
 * connection face is connected to Element 2 in Zone 1 and the
 * remaining connections are Element 3 in Zone 1.
 */
INTEGER4 FaceBndryConnectionElems_Z2[3]  = {2, 3, 3};
INTEGER4 FaceBndryConnectionZones_Z2[3]  = {1, 1, 1};

I = TECPOLYBCONN142(&TotalNumBndryFaces_Z2,
```

```
                    FaceBndryConnectionCounts_Z2,
                    FaceBndryConnectionElems_Z2,
                    FaceBndryConnectionZones_Z2);
```

Step 12 Close the file using TECEND

Call `TECEND` to close the file.

```
I = TECEND142();
```

Polyhedral Example

The following example (written in C) illustrates how to create a single polyhedral cell using the TecIO library.

```
#include "TECIO.h"
#include "MASTER.h" /* for defintion of NULL */

int main()
{
 /* Call TECINI142 */
 INTEGER4 FileType   = 0;  /* 0 for full file  */
 INTEGER4 FileFormat = 0; // 0 == PLT, 1 == SZPLT; Only PLT is currently
                          // supported for polyhedral zones
 INTEGER4 Debug      = 0;
 INTEGER4 VIsDouble  = 1;
 INTEGER4 I          = 0;  /* use to check return codes */

 I = TECINI142((char*)"Pyramid",      /* Data Set Title  */
               (char*)"X Y Z",        /* Variable List  */
               (char*)"pyramid.plt",  /* File Name  */
               (char*)".",            /* Scratch Directory  */
               &FileFormat,
               &(FileType),
               &(Debug),
               &(VIsDouble));

 /* Call TECPOLYZNE142 */
 INTEGER4  ZoneType  = 7;  /* 7 for FEPolyhedron */
 INTEGER4  NumNodes  = 5;  /* number of unique nodes */
 INTEGER4  NumElems  = 1;  /* number of elements */
 INTEGER8  NumFaces  = 5;  /* number of unique faces */

 double  SolTime     = 12.65;  /* solution time  */
 INTEGER4  StrandID  = 0;      /* static zone  */
```

```
INTEGER4  unused    = 0;      // ParentZone is no longer used

INTEGER4 *PassiveVarArray = NULL;
INTEGER4 *ValueLocArray   = NULL;
INTEGER4 *VarShareArray   = NULL;
INTEGER4  ShrConn         = 0;

/* The number of face nodes in the zone.  This example creates
 * a zone with a single pyramidal cell.  This cell has four
 * triangular faces and one rectangular face, yielding a total
 * of 16 face nodes.
 */
INTEGER8  NumFaceNodes  = 16;
INTEGER4  NumBConns     = 0;  /* No Boundary Connections */
INTEGER4  NumBItems     = 0;  /* No Boundary Items */

I = TECPOLYZNE142((char*)"Polyhedral Zone (Octahedron)",
                  &ZoneType,
                  &NumNodes,
                  &NumElems,
                  &NumFaces,
                  &NumFaceNodes,
                  &SolTime,
                  &StrandID,
                  &unused,
                  &NumBConns,
                  &NumBItems,
                  PassiveVarArray,
                  ValueLocArray,
                  VarShareArray,
                  &ShrConn);

/* Initialize arrays of nodal data */
double *X = new double[NumNodes];
double *Y = new double[NumNodes];
double *Z = new double[NumNodes];

X[0] = 0;
Y[0] = 0;
Z[0] = 0;

X[1] = 1;
Y[1] = 1;
Z[1] = 2;

X[2] = 2;
Y[2] = 0;
Z[2] = 0;
```

```
X[3] = 2;
Y[3] = 2;
Z[3] = 0;

X[4] = 0;
Y[4] = 2;
Z[4] = 0;

/* Write the data (using TECDAT142) */
INTEGER4 DIsDouble = 1;  /* One for double precision */
I = TECDAT142(&NumNodes, X, &DIsDouble);
I = TECDAT142(&NumNodes, Y, &DIsDouble);
I = TECDAT142(&NumNodes, Z, &DIsDouble);

delete X;
delete Y;
delete Z;

/* Define the Face Nodes.
 *
 * The FaceNodes array is used to indicate which nodes define
 * which face. As mentioned earlier, the number of the nodes is
 * implicitly defined by the order in which the nodal data is
 * provided.  The first value of each nodal variable describes
 * node 1, the second value describes node 2, and so on.
 *
 * The face numbering is implicitly defined.  Because there are
 * two nodes in each face, the first two nodes provided define
 * face 1, the next two define face 2 and so on.  If there was
 * a variable number of nodes used to define the faces, the
 * array would be more complicated.
 */

INTEGER4  *FaceNodeCounts = new INTEGER4[NumFaces];
/* The first four faces are triangular, i.e. have three nodes.
 * The fifth face is rectangular, i.e. has four nodes. */
FaceNodeCounts[0] = 3;
FaceNodeCounts[1] = 3;
FaceNodeCounts[2] = 3;
FaceNodeCounts[3] = 3;
FaceNodeCounts[4] = 4;

INTEGER4  *FaceNodes = new INTEGER4[NumFaceNodes];
/* Face Nodes for Face 1 */
FaceNodes[0]  = 1;
FaceNodes[1]  = 2;
FaceNodes[2]  = 3;
```

```
/* Face Nodes for Face 2 */
FaceNodes[3]  = 3;
FaceNodes[4]  = 2;
FaceNodes[5]  = 4;

/* Face Nodes for Face 3 */
FaceNodes[6]  = 5;
FaceNodes[7]  = 2;
FaceNodes[8]  = 4;

/* Face Nodes for Face 4 */
FaceNodes[9]  = 1;
FaceNodes[10] = 2;
FaceNodes[11] = 5;

/* Face Nodes for Face 5 */
FaceNodes[12] = 1;
FaceNodes[13] = 5;
FaceNodes[14] = 4;
FaceNodes[15] = 3;

/* Define the right and left elements of each face.
 *
 * The last step for writing out the polyhedral data is to
 * define the right and left neighboring elements for each
 * face.  The neighboring elements can be determined using the
 * right-hand rule.  For each face, place your right-hand along
 * the face which your fingers pointing the direction of
 * incrementing node numbers (i.e. from node 1 to node 2).
 * Your right thumb will point towards the right element; the
 * element on the other side of your hand is the left element.
 *
 * The number zero is used to indicate that there isn't an
 * element on that side of the face.
 *
 * Because of the way we numbered the nodes and faces, the
 * right element for every face is the element itself
 * (element 1) and the left element is "no-neighboring element"
 * (element 0).
 */

INTEGER4  *FaceLeftElems  = new INTEGER4[NumFaces];
FaceLeftElems[0]  = 1;
FaceLeftElems[1]  = 1;
FaceLeftElems[2]  = 0;
FaceLeftElems[3]  = 0;
FaceLeftElems[4]  = 0;
```

```
 INTEGER4  *FaceRightElems = new INTEGER4[NumFaces];
 FaceRightElems[0]  = 0;
 FaceRightElems[1]  = 0;
 FaceRightElems[2]  = 1;
 FaceRightElems[3]  = 1;
 FaceRightElems[4]  = 1;

 /* Write the face map (created above) using TECPOLYFACE142. */
 INTEGER4 NumFaces32 = (INTEGER4)NumFaces;
 I = TECPOLYFACE142(&NumFaces32,
 FaceNodeCounts,  /* The face node counts array */
 FaceNodes,       /* The face nodes array */
 FaceLeftElems,   /* The left elements array  */
 FaceRightElems); /* The right elements array  */

 delete FaceNodeCounts;
 delete FaceNodes;
 delete FaceLeftElems;
 delete FaceRightElems;

 I = TECEND142();
 return 0;
}
```

IJ-ordered zone

The following example illustrates how to create a simple IJ-ordered zone. `TECZNE142` is called first to initialize the zone.

```
// Internal testing flags
// RUNFLAGS:none
// RUNFLAGS:--szl


#include "TECIO.h"
#include "MASTER.h" /* for defintion of NULL */
#include <string.h>

int main(int argc, const char *argv[])
{
    INTEGER4 Debug      = 1;
    INTEGER4 VIsDouble  = 0;
    INTEGER4 FileType   = 0;
    INTEGER4 fileFormat; // 0 == PLT, 1 == SZPLT
    if (argc == 2 && strncmp(argv[1],"--szl",5) == 0)
```

```
   fileFormat = 1;
else
   fileFormat = 0;

INTEGER4 I  = 0; /* Used to track return codes */

/*
 * Open the file and write the tecplot datafile
 * header information
 */
I = TECINI142((char*)"IJ Ordered Zones", /* Name of the entire
                                          * dataset.
                                          */
              (char*)"X Y P",  /* Defines the variables for the data
                                * file. Each zone must contain each of
                                * the variables listed here. The
                                * order of the variables in the list is
                                * used to define the variable number
                                * e.g.  X is Var 1).
                                */
              (char*)"ij_ordered.plt",
              (char*)".",  /* Scratch Directory */
              &fileFormat,
              &FileType,
              &Debug,
              &VIsDouble);

float X1[4];
float Y1[4];
float P1[4];
float X2[4];
float Y2[4];
float P2[4];

INTEGER4 ICellMax  = 0;
INTEGER4 JCellMax  = 0;
INTEGER4 KCellMax  = 0;
INTEGER4 DIsDouble  = 0;
double  SolTime  = 360.0;
INTEGER4 StrandID  = 0;  /* StaticZone */
INTEGER4 unused  = 0;  // ParentZone is no onger used
INTEGER4 IsBlock  = 1;  /* Block */
INTEGER4 NFConns  = 0;
INTEGER4 FNMode  = 0;
INTEGER4 TotalNumFaceNodes  = 1;
INTEGER4 TotalNumBndryFaces  = 1;
INTEGER4 TotalNumBndryConnections = 1;
INTEGER4 ShrConn  = 0;
```

```
/*Ordered Zone Parameters*/
INTEGER4 IMax = 2;
INTEGER4 JMax = 2;
INTEGER4 KMax = 1;

X1[0] = .125;
Y1[0] = .5;
P1[0] = 5;

X1[1] = .625;
Y1[1] = .5;
P1[1] = 7.5;

X1[2] = .125;
Y1[2] = .875;
P1[2] = 10;

X1[3] = .625;
Y1[3] = .875;
P1[3] = 7.5;

X2[0] = .375;
Y2[0] = .125;
P2[0] = 5;

X2[1] = .875;
Y2[1] = .125;
P2[1] = 7.5;

X2[2] = .375;
Y2[2] = .5;
P2[2] = 10;

X2[3] = .875;
Y2[3] = .5;
P2[3] = 7.5;

/*  Ordered Zone */
INTEGER4 ZoneType = 0;
I = TECZNE142((char*)"Ordered Zone",
              &ZoneType,
              &IMax,
              &JMax,
              &KMax,
              &ICellMax,
              &JCellMax,
              &KCellMax,
```

```
                    &SolTime,
                    &StrandID,
                    &unused,
                    &IsBlock,
                    &NFConns,
                    &FNMode,
                    &TotalNumFaceNodes,
                    &TotalNumBndryFaces,
                    &TotalNumBndryConnections,
                    NULL,
                    NULL,
                    NULL,
                    &ShrConn);
    INTEGER4 III = IMax * JMax * KMax;
    I  = TECDAT142(&III, X1, &DIsDouble);
    I  = TECDAT142(&III, Y1, &DIsDouble);
    I  = TECDAT142(&III, P1, &DIsDouble);

    I = TECZNE142((char*)"Ordered Zone2",
                    &ZoneType,
                    &IMax,
                    &JMax,
                    &KMax,
                    &ICellMax,
                    &JCellMax,
                    &KCellMax,
                    &SolTime,
                    &StrandID,
                    &unused,
                    &IsBlock,
                    &NFConns,
                    &FNMode,
                    &TotalNumFaceNodes,
                    &TotalNumBndryFaces,
                    &TotalNumBndryConnections,
                    NULL,
                    NULL,
                    NULL,
                    &ShrConn);

    I  = TECDAT142(&III, X2, &DIsDouble);
    I  = TECDAT142(&III, Y2, &DIsDouble);
    I  = TECDAT142(&III, P2, &DIsDouble);

    I = TECEND142();
    return 0;
}
```

Switching Between Two Files

In this simplified example, information is written to two separate files. First, one file is created and a zone is written to the file. Then, a second file is created and a zone and auxiliary data are written to the file. The second file is closed and the auxiliary data is written to the first file.

```
#include "TECIO.h"
#include "MASTER.h" /* for defintion of NULL */
#include <string.h>

int main(int argc, const char *argv[])
{
    /*
    * Open the file and write the tecplot datafile
    * header information
    */
    INTEGER4 Debug  = 1;
    INTEGER4 VIsDouble  = 0;
    INTEGER4 FileType  = 0;

    INTEGER4 fileFormat; // 0 == PLT, 1 == SZPLT
    if (argc == 2 && strncmp(argv[1],"--szl",5) == 0)
        fileFormat = 1;
    else
        fileFormat = 0;

    INTEGER4 I  = 0;  /* Used to check the return value */

    I = TECINI142((char*)"SIMPLE DATASET", /* Name of the entire dataset.*/

                  (char*)"X1 Y1 P1",  /* Defines the variables for the data
                                       * file. Each zone must contain each of
                                       * the variables listed here. The order
                                       * of the variables in the list is used
                                       * to define the variable number e.g.
                                       * X1 is Var 1).
                                       */
                  (char*)"multiplefiles-file1.plt",
                  (char*)".",  /* Scratch Directory */
                  &fileFormat,
                  &FileType,
                  &Debug,
                  &VIsDouble);

    /* Set the parameters for TecZne */
    INTEGER4 ZoneType           = 0; /* sets the zone type to ordered */
    INTEGER4 IMax               = 2; /* Create an IJ-ordered zone,
```

```
                                * by using IMax and JMax
                                * values that are greater
                                * than one, and setting KMax
                                * to one.  */
INTEGER4 JMax               = 2;
INTEGER4 KMax               = 1;

double  SolTime             = 0;
INTEGER4 StrandID           = 0; /* StaticZone */
INTEGER4 unused             = 0; // ParentZone is no longer used

INTEGER4 ICellMax           = 0; /* not used */
INTEGER4 JCellMax           = 0; /* not used */
INTEGER4 KCellMax           = 0; /* not used */

INTEGER4 IsBlock            = 1; /* Block */

INTEGER4 NFConns            = 0; /* this example does not use
                                * face neighbors */
INTEGER4 FNMode             = 0;
INTEGER4 TotalNumFaceNodes  = 1;
INTEGER4 TotalNumBndryFaces = 1;
INTEGER4 TotalNumBndryConn  = 1;
INTEGER4 ShrConn            = 0;


/* Create an Ordered Zone */
I = TECZNE142((char*)"Ordered Zone",
              &ZoneType,
              &IMax,
              &JMax,
              &KMax,
              &ICellMax,
              &JCellMax,
              &KCellMax,
              &SolTime,
              &StrandID,
              &unused,
              &IsBlock,
              &NFConns,
              &FNMode,
              &TotalNumFaceNodes,
              &TotalNumBndryFaces,
              &TotalNumBndryConn,
              NULL,
              NULL,
              NULL,
              &ShrConn);
```

```
/* Set the variable values for the ordered zone. */
float X1[4];
float Y1[4];
float P1[4];

X1[0] = 0.125;
Y1[0] = 0.5;
P1[0] = 7.5;

X1[1] = 0.625;
Y1[1] = 0.5;
P1[1] = 10.0;

X1[2] = 0.125;
Y1[2] = 0.875;
P1[2] = 5.0;

X1[3] = 0.625;
Y1[3] = 0.875;
P1[3] = 7.5;

INTEGER4 DIsDouble  = 0;  /* set DIsDouble to 0, for float values. */

INTEGER4 III = IMax * JMax * KMax;
I  = TECDAT142(&III, X1, &DIsDouble);
I  = TECDAT142(&III, Y1, &DIsDouble);
I  = TECDAT142(&III, P1, &DIsDouble);

/* Open a new data file.  note: the first file is still open
* because TecEnd was not called.
*/
I = TECINI142((char*)"Auxiliary Data",
              (char*)"X1 Y1 P1",
              (char*)"multiplefiles-file2.plt",
              (char*)".",
              &fileFormat,
              &FileType,
              &Debug,
              &VIsDouble);

/* Switch the active file to the newly created data file
 * (file2.plt) which is the second file opened with TECINI142
 * so we use 2.
 */
INTEGER4 WhichFile = 2;
I = TECFIL142(&WhichFile);
```

```
    /* Create a second zone, using many of the values from the first
     * zone, and write it to the second data file.
     */

    I = TECZNE142((char*)"Ordered Zone2",
                  &ZoneType,
                  &IMax,
                  &JMax,
                  &KMax,
                  &ICellMax,
                  &JCellMax,
                  &KCellMax,
                  &SolTime,
                  &StrandID,
                  &unused,
                  &IsBlock,
                  &NFConns,
                  &FNMode,
                  &TotalNumFaceNodes,
                  &TotalNumBndryFaces,
                  &TotalNumBndryConn,
                  NULL,
                  NULL,
                  NULL,
                  &ShrConn);
    /* set the variable values for the second zone */
    float X2[4];
    float Y2[4];
    float P2[4];

    X2[0] = 0.375;
    Y2[0] = 0.125;
    P2[0] = 5;

    X2[1] = 0.875;
    Y2[1] = 0.125;
    P2[1] = 7.5;

    X2[2] = 0.375;
    Y2[2] = 0.5;
    P2[2] = 10;

    Y2[3] = 0.5;
    X2[3] = 0.875;
    P2[3] = 7.5;

    III = IMax * JMax * KMax;
    I  = TECDAT142(&III, X2, &DIsDouble);
```

```
    I  = TECDAT142(&III, Y2, &DIsDouble);
    I  = TECDAT142(&III, P2, &DIsDouble);

    /* Switch to the first file. */
    WhichFile = 1;
    I = TECFIL142(&WhichFile);

    /* Create an auxiliary data value and write it to the file */
    char DeformationValue[128];
    strcpy(DeformationValue, "0.98");

    I = TECAUXSTR142((char*)"DeformationValue",
                     DeformationValue);
    /* Close the first file */
    I = TECEND142();

    /* The remaining file will become the active file.  As such,
     * TecFil does not need to be called again to close the second
     * file.
     */
    I = TECEND142();

    return 0;
}
```

Text Example

The following example creates a data file with a single text box reading "Sample Text".

```
#include "TECIO.h"
#include <string.h>

int main(int argc, const char *argv[])
{
    /* Open the file & write the datafile header information */
    INTEGER4 Debug  = 1;
    INTEGER4 VIsDouble = 0;

    INTEGER4 fileFormat; // 0 == PLT, 1 == SZPLT
    if (argc == 2 && strncmp(argv[1],"--szl",5) == 0)
        fileFormat = 1;
    else
        fileFormat = 0;

    INTEGER4 FileType  = 0;
    INTEGER4 I         = 0;  /* used to check the return value */
```

```
I = TECINI142((char*)"Text",
              (char*)"X Y P",
              (char*)"text.plt",
              (char*)".",
              &fileFormat,
              &FileType,
              &Debug,
              &VIsDouble);

/* Specify the X, Y and Z position of the anchor point */
double  XPos  = 0.0;
double  YPos  = 1.0;
double  ZPos  = 0.0; /* N/A for 2D text */

INTEGER4 PosCoordMode  = 0;  /* use grid coordinates */

/* opt not to attach the text to a given zone.  When text is
 * attached to a given zone, it is displayed only when the zone
 * is displayed.
 */
INTEGER4 AttachToZone  = 0;
INTEGER4 Zone          = 2;


/* Specify the font values */
INTEGER4 Font             = 1;  /* Helvetica Bold */
INTEGER4 FontHeightUnits  = 2;  /* in grid coordinates */
double  FontHeight        = 18;

/* Set the box style parameters */
INTEGER4 BoxType          = 1;  /* filled box */
double  BoxMargin         = .5; /* margin between the text
                                 * and the text box */
double  BoxLineThickness  = .1;
INTEGER4 BoxColor         = 0;  /* set the box line color to black.*/
INTEGER4 BoxFillColor     = 1;  /* set the box fill color to red.*/

/* set the font properties */
double  Angle             = 30; /* angle of the text */
INTEGER4 Anchor           = 1;  /* set the anchor point to
                                 * the center of the text box.
                                 */
double  LineSpacing       = 1.5;
INTEGER4 TextColor        = 7;  /* set the font color to white /*

INTEGER4 Scope            = 1;  /* set the text to "local",
                                 * i.e. available in the
```

```
                                    * current frame only. */
    INTEGER4 Clipping         = 1;


    char  Text[60];
    char  MFC[24];
    strcpy(Text, "Sample Text");
    strcpy(MFC, "My Macro");

    I = TECTXT142(&XPos,
                  &YPos,
                  &ZPos,
                  &PosCoordMode,
                  &AttachToZone,
                  &Zone,
                  &Font,
                  &FontHeightUnits,
                  &FontHeight,
                  &BoxType,
                  &BoxMargin,
                  &BoxLineThickness,
                  &BoxColor,
                  &BoxFillColor,
                  &Angle,
                  &Anchor,
                  &LineSpacing,
                  &TextColor,
                  &Scope,
                  &Clipping,
                  Text,
                  MFC);

    I = TECEND142();

    return 0;
}
```

Partitioned Data Examples

The **brickpartitioned** example supplied in the installation uses the partitioned data writing capability of TecIO to write a FE brick zone in three partitions. It can be used either with the standard TecIO library or with TecIO-MPI (by adding `#define MPI 1` to the top of the file and running it using `mpiexec`). Unlike the other examples, it is written in C++ (a FORTRAN 90 version is also provided). If you have followed the other examples, then you will find the data-writing in this example to be straightforward, and the unfamiliar code will be that dealing with the partition-related function calls: `TECMPIINIT142`, `TECZNEMAP142`, and `TECFEPTN142`.

The following notes may help you find your way around:

- Add `#define MPI 1` to the top of the file to use the MPI version, and link against `libtecoimpi.dll`, `libteciompi.so`, or `libteciompi.dylib` as appropriate to your platform.
- In MPI mode, the `MPI_COMM_WORLD` communicator is used, and the main rank is designated as 0 (the first rank), both of which should be typical.
- The zone is partitioned using pseudo-IJK indices. This is for demonstration purposes only; you will likely need a different strategy in real solver code. The partitioning is defined (along with the data) in the function `createData`.

The **ijkpartitioned** example in the installation writes IJK-indexed data using the TecIO-MPI API. Similarly to the **brickpartitioned** example, the main additions are the partition-related calls: `TECMPIINIT142`, `TECZNEMAP142`, and in this case `TECIJKPTN142`.

ASCII Data

Files exported into Tecplot's data format may be either ASCII or binary. However, we strongly recommend using Tecplot's binary file format (*.plt). The ASCII file format is provided to illustrate how data is structured in Tecplot. ASCII data format is useful only for very small data files. Reading an ASCII data file into Tecplot 360 can be much slower than reading a binary data file, as binary data files are structured for more efficient data access, and Tecplot 360 must convert from ASCII to binary prior to loading the data. Refer to Binary Data for information on creating files in Tecplot's binary format.

Converting ASCII to Binary

Tecplot 360 or Preplot can be used to convert ASCII data files to binary as part of a post-processing step, before the data is presented for interactive visualization. See the User's Manual for converting with Tecplot 360, or ASCII Data File Conversion to Binary for converting with Preplot. A description of the binary format is included in Binary Data File Format. If your data is generated in FORTRAN or C, you may be able to generate binary data files directly using the utilities described in Binary Data.

Syntax Rules & Limits

An ASCII data file begins with a file header defining a title for the data file and/or the names of the variables. The header is followed by 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. The records in the file may be in any order.

ASCII data files have the following limits:

Number of Records

Each data file may have ten custom label records, and any number of text and geometry records.

Maximum Characters per Line

The maximum length of a line in a data file is 32,000 characters.

There are additional limits specific to some of the record types and parameters. These limits are discussed in the section for the associated record type or parameter.

When writing an ASCII data file, please keep the following syntax rules in mind:

Character Strings

Double quotes must be used to enclose character strings with embedded blank spaces or other special characters.

Multiple Lines

Any line may be continued onto one or more following lines (except for text enclosed in double quotes ").

Escape Characters

A backslash \ may be used to remove the significance of (or escape) the next character (that is, \" produces a single double-quote).

Comments

Any line beginning with an # is treated as a comment and ignored.

The following simple example of a Tecplot 360 ASCII data file has one small zone and a single line of text:

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

ASCII File Structure

An ASCII data file begins with an 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. Refer to the User's Manual for a complete description of ordered and finite element data. You may also include text, geometry, and custom-label records, in any order.

The first line in a zone, text, geometry, custom label, data set auxiliary data record or variable auxiliary

record begins with the keyword ZONE, TEXT, GEOMETRY, CUSTOMLABELS, DATASETAUXDATA, or VARAUXDATA.

The primary components of ASCII data files are:

- File Header
- Zone Record
- Text Record
- Geometry Record
- Custom Labels Record
- Data Set Auxiliary Data Record
- Variable Auxiliary Data Record

File Header

The File Header is an optional component of an ASCII data file. It may contain a TITLE, FILETYPE and/or a VARIABLES list. 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 360 (provided each data file has the same number of variables per data point).

File Header Components

Token	Syntax	Notes
TITLE	= "<*string*>"	The title will be displayed in the headers of Tecplot 360 frames.
FILETYPE	=FULL, GRID or SOLUTION	Specifies the data file type. A full data file contains both grid and solution data. If omitted, the FILETYPE will be treated as "FULL".
VARIABLES	= "*VARNAME1*" , "*VARNAME2*", "*VARNAME3*", ..., "*VARNAMEN*"	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. Tecplot 360 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 360 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 360 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. However, you may 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 via the **Select Variables** dialog (accessed via **Plot>Assign XYZ**).

Example Grid File

The following example displays a very simple 2D grid file.

```
#"Grid" files look like standard Tecplot data files with no solution variables.
TITLE = "Example Grid File"
FILETYPE = GRID
VARIABLES = "X" "Y"
ZONE
I = 3, J = 3, K = 1
ZONETYPE = Ordered, DATAPACKING = BLOCK
0.0  0.5   1.0   0.0   0.5   1.0   0.0   0.5   1.0
0.0  0.0   0.0   0.5   0.5   0.5   1.0   1.0   1.0
```

Example Solution File

The following example displays a very simple solution file (to be used with the Example Grid File).

```
TITLE     = "Example Solution File"
FILETYPE  = SOLUTION
VARIABLES = "Pressure"
ZONE
I = 3, J = 3, K = 1
ZONETYPE = Ordered, DATAPACKING = BLOCK
2.0   2.0   2.0   0.0   0.0   0.0   2.0   2.0   2.0
```

Zone Record

A zone record consists of a control line that begins with the keyword ZONE, followed by the zone header, followed by a set of numerical data called the zone data. The contents of the zone footer depend upon the type of zone. Refer to the following table for an overview of the contents of a zone record.

Component	Notes
ZONE	The keyword "ZONE" is required at the start of every zone record
Zone Header	The Zone Header is used to specify the type of data in the zone, the structure of the data, the names of the variables in the zone, and more. Refer to Zone Header for details.
Data	The data section follows the zone header. The arrangement of the data is dependent upon the values of DATA PACKING and VAR LOCATION (specified in the Zone Header). Refer to Data for details.

Component	Notes
Zone Footer	The contents required for the Zone Footer depend upon the ZONETYPE (specified in the Zone Header). For ordered zones, the Zone Footer contains the Face Neighbor Connections List information (if any). For cell-based finite element zones (`FETRIANGLE`, `FEQUADRILATERAL`, `FETETRAHEDRAL` and `FEBRICK`), the Zone Footer contains Connectivity information, followed by Face Neighbor Connections List. For face-based finite element zones (`FEPOLYHEDRAL`, `FEPOLYGON`), the Zone Footer contains Facemap Data followed by Boundary Map Data. Remember that Tecplot Focus cannot load this type of zone. Refer to Zone Footer for additional information.

Zone Header

Keyword	Syntax	Required (Y/N)	Default	Notes
ZONE		Y		Keyword required to start a zone record
T	= <*string*>	N		Zone Title. This may be any text string up to 128 characters in length. If you supply a longer text string, it is automatically truncated to the first 128 characters. The titles of zones appear in the **Zone Style** and other dialogs, and, optionally, in the XY- plot legend.

Keyword	Syntax	Required (Y/N)	Default	Notes
ZONETYPE	= <zonetype>	N	ORDERED	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 is presumed if the ZONETYPE parameter is omitted. See Ordered Data for more information on ordered zones, and Finite Element Data for details on finite element data. When specifying your ZoneType, please note that Tecplot Focus cannot read FEPOLYGON or FEPOLYHEDRON zones.
I	= <*integer*>	Y		Specify the maximum number of points in the I- J- or K-direction. Use only when ZONETYPE is ORDERED.
J	= <*integer*>	Y		
K	= <*integer*>	Y		
NODES	= <integer>	Y		Use for finite element zone types only (that is, not ordered zones). Specify the total number of NODES and ELEMENTS in the data file. Refer to Finite Element Data for additional information.
ELEMENTS	= <*integer*>	Y		
FACES	= <*integer*>	Y (for face-based finite element zones)		Use for face-based finite element zones types (polygonal and polyhedral) only. Specify the number of FACES in the data file. See Finite Element Data for more information.

Keyword	Syntax	Required (Y/N)	Default	Notes
TOTALNUM FACENODES	= <*integer*>	Y (for polyhedral zones)		For face-based finite element zones only. Total number of nodes in the Facemap Data section for all faces. This is optional for polygons as TotalNumFaceNodes = 2*NumFaces.
NUM CONNECTED BOUNDARY FACES	= <*integer*>	Y		For face-based finite element zones only. Total number of boundary faces listed in the Facemap Data section. Set to zero if boundary faces aren't used.
TOTALNUM BOUNDARY CONNECTIONS	= <*integer*>	Y		For face-based finite element zones only. Total number of entries for boundary items listed in the Facemap Data section. Set to zero if boundary faces aren't used.
FACENEIGH BORMODE	= *[*LOCALONE TOONE, LOCALONE TOMANY, GLOBALONE TOONE, GLOBALONE TOMANY*]*	N	LOCAL ONETOONE	For ordered or cell-based finite element zones only. Used to indicate whether the neighboring faces are within the current zone or in another zone (i.e. local or global), as well as whether the connections are one-to-one or one-to-many. When this token is used, both the FACENEIGHBORCONNECTIONS token and the FaceNeighbor Connections List are required. Refer to Face Neighbor Connections List for details.

Keyword	Syntax	Required (Y/N)	Default	Notes
FACE NEIGHBOR CONNECTIONS	= <*integer*>	Y, if FACE NEIGHBOR MODE, is in use.		For ordered or cell-based finite element zones only. Used to indicate the total number of connections for all elements in the zone. For example, if you have two cells with three connections each, the number of face neighbor connections is equal to six. When this token is used, both the FACENEIGHBORMODE token and the FaceNeighbor Connections List are required. Refer to Face Neighbor Connections List for details.
DT	= (<datatype> *for var1*, <datatype> *for var2 ...*, <datatype> *for varn*)	N	SINGLE	Each variable in each zone in the data file may have its own data type. The data type determines the amount of storage Tecplot 360 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 360 treats numeric data as data type SINGLE. If any variable in the zone uses the BIT data type, the DATA PACKING must be `BLOCK`. Refer to Data for details.

Keyword	Syntax	Required (Y/N)	Default	Notes
DATAPACKING	= <datapacking>	N	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. BLOCK format must be used for cell-centered data and polyhedral zones (FEPOLYGON/FEPOLYHEDRAL), as well as for all binary data.
VAR LOCATION	=([*set-of-vars*] =<varlocation>, [*set-of-vars*] =<varlocation>, ...)	N	NODAL	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. 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.
VAR SHARELIST	=([*set-of-vars*] =<zone>, [*set-of-vars*] =<zone>)	N	If zone number is omitted, the variables are shared from the previous zone.	Used for variables that are exactly the same for a set of zones. Specify the integer value of the source zone. 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.

Keyword	Syntax	Required (Y/N)	Default	Notes
NV	= <*integer*>	N		Specifies the variable number of the variable representing the "Node" value in finite element data. The NV parameter is used infrequently. It is mostly used when the order in which nodes are listed in the data file does not match the node numbering desired in the plot. Refer to Finite Element Zone Node Variable Parameters Example for an example using the NV parameter.
CONNECTIVITY SHAREZONE	=<zone>	N		Specify the number of the zone from which the connectivity is shared. The connectivity list (cell-basedfinite element only) and face-neighbors may be shared between zones using the CONNECTIVITYSHAREZONE parameter in the control line of the current zone. Alternatively, the parameter may be used to share the Facemap Data for face-based finite element zones.To use connectivity sharing, the zone must have the same number of points and elements (and faces, if the zone is face-based), and be the same zone type.

Keyword	Syntax	Required (Y/N)	Default	Notes
STRANDID	= <*integer*>	N		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.
SOLUTIONTIME	= <*double*>	N		Specify a floating point time value representing the solution time. Zones can be organized together by associating themselves to the same STRANDID.
PASSIVEVARLIST	= [*set-of-vars*]	N	All variables non-passive	Use this option to make variables passive. For example, to make variables 4, 5, and 20 passive, use this syntax: `PASSIVEVARLIST=[4-5,20]` See Best Practices, [introduction/best-practices/passive-variables], for information on passive variables.

Keyword	Syntax	Required (Y/N)	Default	Notes
AUXDATA	NAME = <*string*>	N		Auxiliary data strings associated with the current zone are specified with the AUXDATA parameter in the control line. This auxiliary data may be used in dynamic text, equations, macros, or add-ons. There may be multiple AUXDATA parameters in the control line for a zone, but names must be unique. NOTE: The NAME portion of the string cannot contain spaces. Auxiliary data is provided as named strings: `AUXDATA EXPERIMENTDATE ="October 13, 2007, 8 A.M."`

Data

Tecplot 360 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 zero to 255).
- `BIT`

The arrangement of ASCII data depends upon the combination of datapacking (BLOCK or POINT), variable location (NODAL or CELL-CENTERED). The zone type also plays a role in that not all forms of datapacking and variable locations are supported by all zone types. In BLOCK data, the data is arranged by variable, while in POINT data the data is arranged by point (node or data point, depending upon the zone type). In NODAL data the variable values are defined at every node (FE data) or point (ORDERED data). In CELLCENTERED data, the variable values are defined at the center of every cell (ORDERED data) or element (FE data).

The available combinations of datapacking and variable location parameters are:

- Block - Nodal

- Block - Cell-centered
- Point - Nodal

The combination of POINT and CELLCENTERED is not available.

BLOCK - NODAL

In block data with nodal values, the data is arranged by variable and each variable is defined at the nodes. The data arrangement is as follows:

A_{11} A_{12} ... A_{1P}

A_{21} A_{22} ... A_{1P}

.....

A_{V1} A_{V2} ... A_{VP}

where:

V = total number of nonpassive, nonshared variables

P = I * J * K *(ordered zones)* or NODES *(FE zones)*

BLOCK - CELLCENTERED

In block data with cell-centered values, the data is arranged by variable and each variable is defined at the center of each cell (ORDERED data) or element (FE data). The data arrangement is as follows:

A_{11} A_{12} ... A_{1P}

A_{21} A_{22} ... A_{1P}

.....

A_{V1} A_{V2} ... A_{VP}

where:

V = total number of nonpassive, nonshared variables

P = (I-1) * (J-1) * (K-1) *(ordered zones[1])*

or

P = ELEMENTS *(FE zones)*

1. For all I, J and K greater than one. When I, J or K is equal to one, a value of one is used instead of subtracting one.

POINT - NODAL

In point data, the values for all variables are given for the first point, then the second point and so on. The variable location is always NODAL.

A_{11} A_{12} ... A_{1V}

A_{21} A_{22} ... A_{1V}

.....

A_{P1} A_{P2} ... A_{PV}

where:

V = total number of nonpassive, nonshared variables

P = I * J * K *(ordered zones)*

or

P = NODES *(FE zones)*

General Formatting Rules

The following formatting guidelines apply to all data arrangements:

- Numerical values in zone data must be separated by one or more spaces, commas, tabs, new lines, or carriage returns. The radix (decimal point) is '.' (period) even in locales that use another character, such as ',' (comma).
- 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
```

Variable Sharing

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 360'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 four, five, six and 11 are shared from zone three, variables 20, 21, 22, and 23 are shared from zone one, 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).

Zone Footer

The contents required for the Zone Footer depend upon the ZONETYPE (specified in the Zone Header.

Ordered zones

The Zone Footer contains the Face Neighbor Connections List (if any).

Cell-based finite element zones (FETRIANGLE, FEQUADRILATERAL, FETETRAHEDRAL and FEBRICK)

the Zone Footer contains Connectivity information, followed by Face Neighbor Connections List (if any).

Face-based finite element zones (FEPOLYHEDRAL, FEPOLYGON)

the Zone Footer contains Facemap Data, followed by Boundary Map Data.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Connectivity

For cell-based finite element zones (FETRIANGLE, FEQUADRILATERAL, FETETRAHEDRAL, and FEBRICK), the nodal data is followed by the connectivity information. The connectivity list is not preceded by a token or keyword. It is simply a list of numbers.

The connectivity list details the node numbers of all of the nodes included in each element. When providing the connectivity list, please keep in mind the following guidelines:

- Each row in the connectivity list corresponds to an element, where the first row corresponds to the first element, and so forth.
- The node numbers must be provided in order, either clockwise or counter-clockwise.
- You must provide the same number of nodes as are included in an element. For example, you must provide eight numbers for BRICK elements and three numbers for TRIANGLE elements. If you are using repeated nodes, provide the node number of the repeated node multiple times.

See also: Connectivity Sharing

The connectivity for face-based zones (FEPOLYGON and FEPOLYHEDRAL) is defined by the Facemap

Data.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

Face Neighbor Connections List

For ordered zones, the data section may be followed with face neighbor connections. For cell-based finite element zones, the data section and connectivity list may be followed by the face neighbor connection information.

Use face neighbors to specify connections between zones (global connections) or connections within zones (local connections). Face neighbor connections are used by Tecplot when deriving variables or drawing contour lines. Specifying face neighbors, typically leads to smoother connections. NOTE: face neighbors have expensive performance implications. Use face neighbors to manually specify connections that are not defined via the connectivity list.

Face neighbor connections are defined by the FACENEIGHBORMODE and FACENEIGHBORCONNECTIONS tokens along with the Face Neighbor Connections list. The `FACENEIGHBORMODE` token is used to specify the type of face neighbor connection used. The `FACENEIGHBORCONNECTIONS` token is used to define the total number of face neighbor connections included in the zone.

The nature of the data arranged in the Face Neighbor Connections 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 or more other cell faces, use `ONETOMANY`

Mode	Number of Values	Order of Data in the Face Neighbor Connections List
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` -the cell number in the current zone
- `fz` - the number of the cell face in the current zone
- `nc` -the cell number of the neighbor cell in the current zone
- `oz` - face obscuration flag (zero for face partially obscured, one for face entirely obscured)
- `nz` - the number of neighboring cells for the `ONETOMANY` options

- `ncn` - the number of the *nth* local zone neighboring cell in the list
- `zr` - the remote zone number
- `cr` - the cell number of the neighboring cell in the remote zone
- `zrn` - the zone number of the nth neighboring cell in the `GLOBALONETOMANY` list
- `crn` - 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 the figures below:

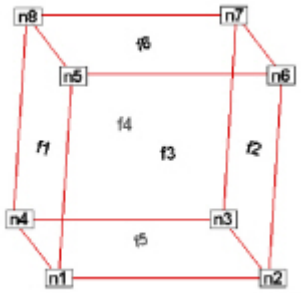


Figure 14. Example of node and face neighbors for an FE-brick cell or IJK-ordered cell.

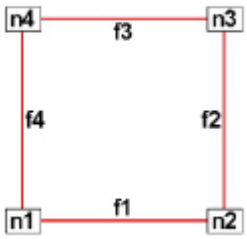


Figure 15. Example of node and face numbering for an IJ-ordered/ FE-quadrilateral cell.

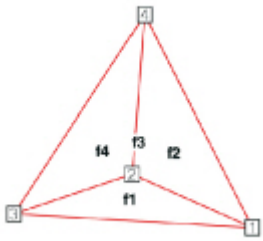


Figure 16. Example of tetrahedron face neighbors.

A connection must be specified for two matching cell faces to be effective. The nature of the Face Neighbor Connections list depends upon its `FACENEIGHBORMODE`.

For example, for data with a `FACENEIGHBORMODE` of `GLOBALONETOONE`, if cell six, face two in zone nine should

be connected to cell one, face four in zone 10, the connections for zone nine 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 (cell#, face#, connecting zone#, connecting cell#)
```

Global face neighbors are useful for telling Tecplot 360 about the connections between zones. This could be used, for example, to smooth out the crease in smooth shading at zone boundaries. For cell-centered data, they can make contours and streamtraces more continuous at zone boundaries.

Facemap Data

For face-based finite element zones (FEPOLYGON and FEPOLYHEDRAL), the data section is followed by the Facemap Data section. If boundary faces are used, the Facemap Data section is followed by the Boundary Map Data data section. Otherwise, the facemap data section marks the end of the zone record.

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

The face map data (in four major groupings) is defined by the following list:

Like the Data section of the zone record, the data region of the Face Map section does not include tokens. It includes a list of data. The descriptors TotalNodesInFace, WhichNodesInFace, LeftNeighborForFace and RightNeighborForFace should not be included in your data file.

TotalNodesInFace

A space-separated list of the total number of nodes in each face:

$NodesInFace_1$ $NodesInFace_2$ ... $NodesInFace_F$

where F is equal to the total number of faces.

The TotalNodesInFace section is not used for polygonal zones, as each face of a polygon always has two nodes.

WhichNodesInFace

A list of the node numbers for each node in each face. The total amount placed on one line.

$Face_1Node_1$ $Face_1Node_2$ ... $Face_1Node_{TotalNodesInFace1}$ $Face_2Node_1$ $Face_2Node_2$ ... $Face_FNode_{TotalNodesInFaceF}$

LeftNeighborForFace

A list of left neighboring elements for each face:

$LeftElementForFace_1$ $LeftElementForFace_2$ ... $LeftElementForFace_F$

RightNeighborForFace

A list of right neighboring elements for each face:

$LeftElementForFace_1$ $LeftElementForFace_2$ ... $LeftElementForFace_F$

The face map may be shared between zones in the same file by specifying the zone number of the sharing zone in place of the CONNECTIVITYSHAREZONE value.

Defining Neighboring Elements

The left element and right element are determined by the left-hand versus right-hand winding rule. The left and right neighboring elements represent elements within the current zone, and they are always "one-to-one". That is, each face represents a complete interface between two elements. A negative value (-t) in either of the neighboring faces lists indicates that the neighboring element(s) are defined in the boundary face section at the t^{th} boundary face. Refer to Boundary Map Data for details.

Any face that has no neighboring element for either its right or left adjacent element, will use a value of zero for the element value.

See also Connectivity Sharing.

Boundary Map Data

If the NUMCONNECTEDBOUNDARYFACES is greater than zero, the boundary map data section is required. The boundary map data section should immediately follow the Facemap Data section. This section does not need to be "one-to-one". One face can link up to multiple elements in other zones.

The boundary map data section need only be used for global or one-to-many connections. Local, one-to-one connections should be defined as left/right elements.

The number of adjacent elements is listed for each of the boundary faces. Then each boundary face lists the element number for each of its adjacent elements. Then each boundary face lists the zone number for each of its adjacent elements (0 can be used to refer to the current zone). The number of the face is not specified but is implicit (first face listed is 1 and corresponds to -1 in the left/right neighbor list, the second is 2 and corresponds to -2, etc.).

Connectivity Sharing

The connectivity list and face neighbor connections (for cell-based finite element zones) or the facemap data (for face-based finite element zones) may be shared between zones by 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.

Text Record

Text records are used to import text directly from a data file. Text can also be imported into Tecplot 360 using a macro file. You may create data files containing only text records and read them into Tecplot 360 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:

Text Record

Token	Syntax	Required (Y/N)	Default	Notes
TEXT		Y		Keyword required to start a text record
T	= <*string*>	Y		The text string is defined in the required `T` (text) parameter. To include multiple lines of text in a single text record, include `\n` in the text string to indicate a new line.
ZN	= <*integer*>	N		Use the `ZN` (*zone*) parameter to attach text to a specific zone or XY mapping. For further information, see the User's Manual.
X	= <*double*>	Y		Specify the x-origin, y-origin and z-origin of the object. The x-origin and y-origin should be in CS (coordinatesys) units. The z-origin of object must always in GRID units.
Y	= <*double*>	Y		
Z	= <*double*>	Y		
R	= <*double*>	Y		r-origin (in CS units) of the object)
THETA	= <*double*>	Y		theta-origin (in CS units) of the object)
CS	= <coordinatesys>	N	FRAME	Text coordinate system. If you specify the frame coordinate system, 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). Specify X, Y and Z for `GRID3D` coordinates. For Polar Line plots, you may specify THETA and R instead of X and Y.

Token	Syntax	Required (Y/N)	Default	Notes
A	= <*double*>	N		Use the A parameter to rotate the text box at an angle counter-clockwise from horizontal. The angle is in units of degrees.
S	= <scope>	N		Scope of the text box. GLOBAL scope attaches the text box to all frames using the same data set. It is the same as selecting the check box Show in "Like" Frames in the Text Options dialog.
BX	= <boxtype>	N	NOBOX	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.
BXF	= <color>	N		BX must be set to FILLED.
BXM	= <*double*>	N		When BX is set to HOLLOW or FILLED, use the BXM token to specify the margin around text in box as fraction of H (text height).
BXO	= <color>	N		When BX is set to HOLLOW or FILLED, use the BXO token to specify the color of the box outline.
LT	= <*double*>	N		When BX is set to HOLLOW or FILLED, use the LT token to specify the thickness of the box outline.
F	=<font>	N		Use the F parameter to specify the font family.
C	= <color>	N		Font Color.
AN	= <textanchor>	N		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 17.
LS	= <*double*>	N	1	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.

Token	Syntax	Required (Y/N)	Default	Notes
H	= <*double*>	N		Specify the height, measured in the units defined by HU.
HU	= <heightunits>	N		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.
MFC	= <*string*>	N		Attach a macro function to the text. The macro function must be a retained macro function that was either set during the current Tecplot session or included in the tecplot.mcr file. Refer to the User's Manual and Scripting Guide for additional information.
CLIPPING	= <clipping>	N		Plot the geometry within to the viewport or the frame.

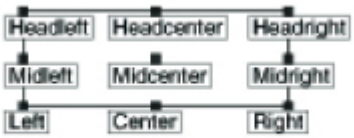


Figure 17. Text anchor positions—values for the AN parameter.

Text Record Examples

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 360.

```
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 360. 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.

Geometry Record Contents

Token	Available Values	Notes
GEOMETRY		Keyword required to start a geometry record
T	= <geomtype>	Geometry Type
F	= <datapacking>	Geometry data format
DT	= <datatype>	Data type
ZN	= <*integer*>	Attach geom to a specific zone or XY mapping. For further information, see the User's Manual.
X	= <*double*>	Specify the x-origin, y-origin and z-origin of the object. The x-origin and y-origin should be in CS (coordinatesys) units. The z-origin of object must always in GRID units.
Y	= <*double*>	
Z	= <*double*>	
R	= <*double*>	r-origin (in CS units) of the object)
THETA	= <*double*>	theta-origin (in CS units) of the object)
CS	= <coordinatesys>	Text coordinate system. If you specify the frame coordinate system, 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). Specify X, Y and Z for GRID3D coordinates. For Polar Line plots, you may specify THETA and R instead of X and Y.
DRAWORDER	= <draworder>	Draw order
S	= <scope>	The S (*scope*) parameter specifies the text scope. GLOBAL scope attaches the text box to all frames using the same data set. It is the same as selecting the check box Show in "Like" Frames in the Geometry Options dialog.

Token	Available Values	Notes
C	= <color>	Geometry outline color.
L	= <linetype>	Line type
PL	= <*double*>	Pattern length (in frame units).
LT	= <*double*>	Line thickness (in frame units)
EP	= <*integer*>	Number of points used to approximate circles or ellipses
FC	= <color>	Fill Color. Any geometry type except LINE3D may be filled with a color by using the FC (fillcolor) parameter. 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.
AST	= <arrowheadstyle>	Arrowhead style
AAT	= <arrowheadattach>	Arrowhead attachment along the line geometry
ASZ	= <*double*>	Size of arrowhead in frame units
AAN	= <*double*>	Angle of arrowhead in degrees
MFC	= <*string*>	You may attach a macro function to the text with the `MFC` parameter. The macro function must be a retained macro function that was either set during the current Tecplot session or included in the `tecplot.mcr` file. Refer to the User's Manual and Scripting Guide for additional information.
CLIPPING	= <clipping>	Plot the geometry within the viewport or the frame.

Data for Geometry Record

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 360 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 polylines, where each polyline can have up to 32,000 points. 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.

Origin positions

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.

Geometry Record Examples

Rectangle

The following geometry record defines a rectangle of 40 width and 30 height:

```
GEOMETRY T=RECTANGLE
40 30 #WIDTH HEIGHT
```

Circle

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  #RADIUS
```

Polyline

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        #Number of polylines
3        #Number of points in polyline 1
0 1      #X, Y coordinates of the point 1 in polyline 1
0 0      #X, Y coordinates of the point 2 in polyline 1
2 0      #X, Y coordinates of the point 3 in polyline 1
2        #Number of points in polyline 2
0 0      #X, Y coordinates of the point 1 in polyline 2
1 2      #X, Y coordinates of the point 2 in polyline 2
```

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

```
GEOMETRY X=50, Y=50, T=LINE, C=CUST3, F=BLOCK, CS=FRAME
2        #Number of polylines
3        #Number of points in polyline 1
0 0 2    #X position of each point in polyline 1
1 0 0    #Y position of each point in polyline 1
2        #Number of points in polyline 2
0 1      #X position of each point in polyline 2
0 2      #y position of each point in polyline 2
```

Ellipse

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         #Horizontal Axis, Vertical Axis
```

3D polyline

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             #Number of polylines
4             #Number of points in polyline 1
0 0 0         #X, Y, Z coordinates of point 1
1 2 2         .
3 2 3         .
```

```
4 1 2          #X, Y, Z coordinates of point 4
```

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

```
GEOMETRY T=LINE3D, F=BLOCK
1              #Number of polylines
4              #Number of points in polyline 1
0 1 3 4        #X position for each point in the polyline
0 2 2 1        #Y position for each point in the polyline
0 2 3 2        #Z position for each point in the polyline
```

Custom Labels Record

The custom label record is an optional record used to provide custom labels for axes, the contour legend or value labels. A single custom label record begins with the keyword CUSTOMLABELS, followed by a series of text strings. The first custom label string corresponds to a value of one on the axis, the next to a value of two, and so forth.

You may have up to ten custom label records in a data file. The custom label set to use is specified via the Tecplot interface. Refer to the User's Manual for details.

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"
```

You may include \n in a custom label to indicate that the following text should start a new line. For example,"Jan\n2012" appears in Tecplot 360 as:

Jan
2012

You must include a data set in order to use custom labels. You cannot use custom labels in files that contain only text and/or geometries.

Data Set Auxiliary Data Record

There is frequently auxiliary data (or Metadata) that helps describe the data set. For example, experimental data may have information about the facility and time at which the data was taken, and other parameters 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.

Auxiliary data are name/value pairs that a user can specify and then use in Tecplot 360 with dynamic text, equations, macros, or add-ons. This data may be with respect to the data set as a whole or it can vary from zone to zone. The ASCII file format token for specifying auxiliary data associated with the entire data set is DATASETAUXDATA, described here. Auxiliary data for a given variable is defined by VARAUXDATA, described in Variable Auxiliary Data Record. Auxiliary data for a given zone is defined by the AUXDATA token within the zone record (refer to Zone Header for details).

The data set auxiliary data control line is as follows:

```
DATASETAUXDATA name = "value"
```

where name is a unique character string with no spaces. You may have multiple `DATASETAUXDATA` records. However, the value of name must be unique for each record.

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 as follows:

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

Similar principles apply when using auxiliary data in text boxes or labels.

Variable Auxiliary Data Record

Variable auxiliary data is added to Tecplot 360 on a per variable basis. Like data set 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. NOTE: The name of an auxiliary data record cannot contain spaces.

ASCII Data File Parameter Assignment Values

The following parameter assignment values are shared among the following types of ASCII file records: Zone Record, Text Record, and/or Geometry Record. Refer to those sections for details.

<arrowheadstyle>	**PLAIN, HOLLOW, FILLED**
<arrowheadattach>	**NONE, BEGINNING, END, BOTH**
<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**
<datatype>	**SINGLE, DOUBLE, LONGINT, SHORTINT, BYTE, BIT**[1]
<draworder>	**AFTERDATA,BEFOREDATA**
<font>	**HELV, HELV-BOLD, TIMES, TIMES-ITALIC, TIMES-BOLD, TIMES-ITALIC-BOLD, COURIER, COURIER-BOLD, GREEK, MATH, USER-DEF**
<geomtype>	**LINE, SQUARE, RECTANGLE, CIRCLE, ELLIPSE, LINE3D**
<heightunits>	**In FRAME coordinatesys either FRAME or POINT; in GRID coordinatesys either GRID or FRAME.**
<linetype>	**SOLID, DASHED, DASHDOT, DOTTED, LONGDASH, DASHDOTDOT**
<scope>	**GLOBAL, LOCAL**
<textanchor>	**LEFT, CENTER, RIGHT, MIDLEFT, MIDCENTER, MIDRIGHT, HEADLEFT, HEADCENTER, HEADRIGHT**
<varlocation>	**NODAL, CELLCENTERED**
<zone>	**zone number to which this item is assigned (0=all)**

<zonetype>	**ORDERED, FELINESEG, FETRIANGLE, FEQUADRILATERAL, FETETRAHEDRON, FEBRICK, FEPOLYGON or FEPOLYHEDRAL. Note that Tecplot Focus cannot load FEPOLYGON or FEPOLYHEDRAL zones.**

Table 3. ASCII Data File Parameter Assignment Values

1. LONGINT, SHORTINT, BYTE, AND BIT are only vailable for Zone Records.

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.

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 360.

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 360 from the zone data if it is not explicitly set by the zone control line (using the `I`-parameter).

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 User's Manual for more information on data structure. In IJ-ordered data, the I-index varies from one 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*.

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 the User's Manual for more information.

In IJK-ordered data, the I-index varies from one 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*.

Ordered Data Examples

The following examples are provided for your reference:

- I-Ordered Data - Simple example
- IJ-Ordered Data - Simple Example
- IJK-Ordered Data - Simple Example
- Multi-Zone XY Line Plot
- Multi-Zone XY Line Plot with Variable Sharing Example
- Cell-Centered Data
- Two-Dimensional Field Plots
- Three-Dimensional Field Plots
- Polygonal - simple example
- Polyhedral - complex example

I-Ordered Data - Simple example

This data set is plotted in Figure 18; each data point is labeled with its I-index.

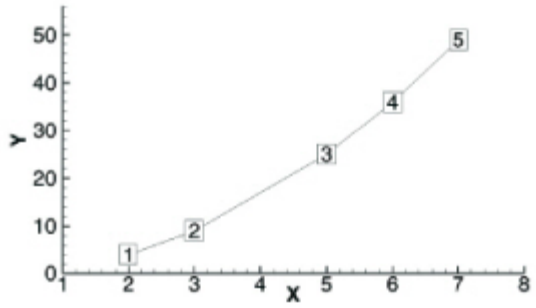


Figure 18. An I-ordered data set.

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 values of each variable are listed, one variable at a time.

FORTRAN Code

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
     WRITE (*,*) ARRAY(VAR,I)
1   CONTINUE
```

IJ-Ordered Data - Simple Example

There are four variables (`X`, `Y`, `Temperature`, `Pressure`) and six data points.

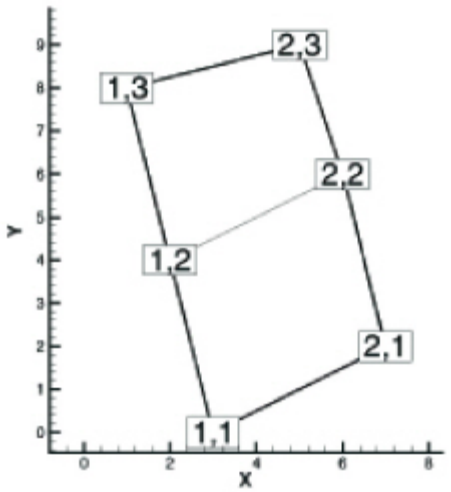


Figure 19. An IJ-ordered data set.

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

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
           WRITE (*,*) ARRAY(VAR,I,J)
1     CONTINUE
```

IJK-Ordered Data - Simple Example

An example of IJK-ordered data in `BLOCK` format is listed below. There are four variables (`X`, `Y`, `Z`, `Temperature`) and twelve data points.This data is plotted in Figure 20; each data point is labeled with its IJK-index.

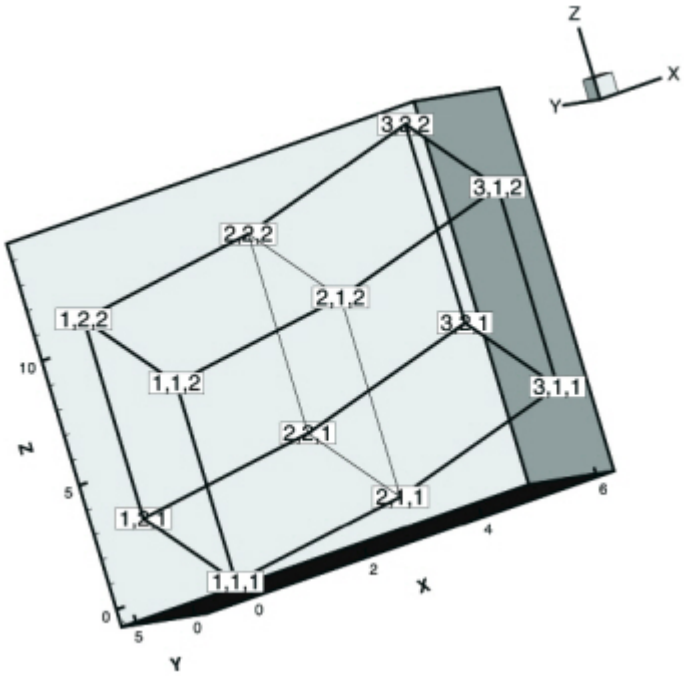


Figure 20. An IJK-ordered data set.

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

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
  WRITE (*,*) ARRAY(VAR,I,J,K)
1   CONTINUE
```

Multi-Zone XY Line Plot

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:				**Time = 0.1 seconds:**		
Position	Temperature	Pressure		Position	Temperature	Pressure
71.30	563.7	101362.5		71.31	564.9	101362.1
86.70	556.7	101349.6		84.42	553.1	101348.9
103.1	540.8	101345.4		103.1	540.5	101344.0
124.4	449.2	101345.2		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 Text Record) to add a title to the plot. The plot shown in Figure 21 can be produced from this file.

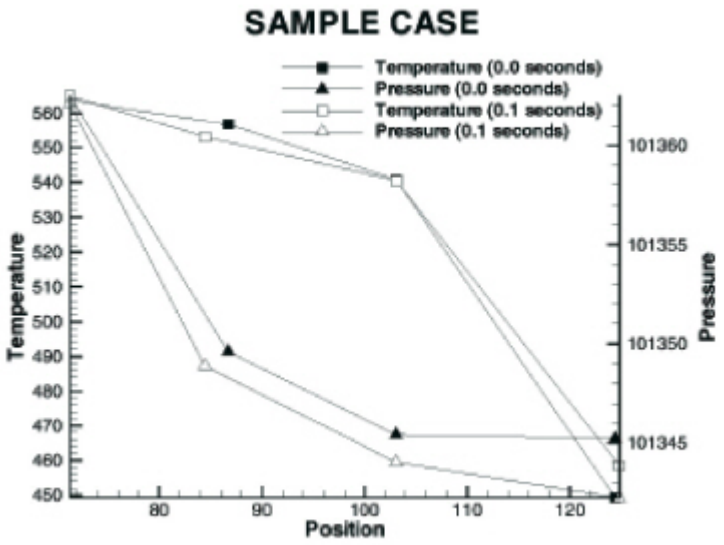


Figure 21. A multi-zone XY Line plot.

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"
```

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)            #share variable 1 from zone 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"
```

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.

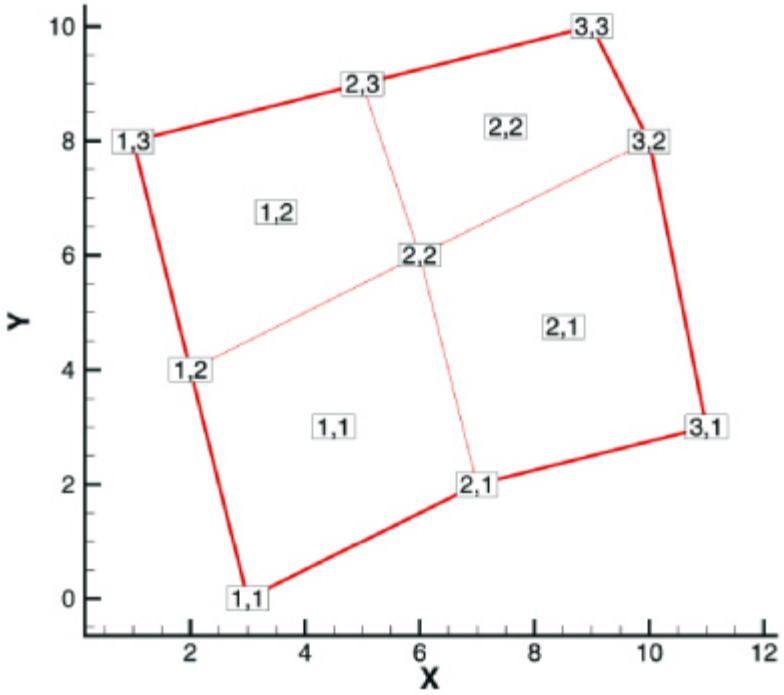


Figure 22. An IJ-ordered data set with cell-centered data.

```
VARIABLES = "X", "Y", "Temperature", "Pressure"
ZONE I=3, J=3, DATAPACKING=BLOCK, VARLOCATION=([3,4]=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 Plot 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. 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 one 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 <FIG:twod-fe-data>>.

Refer to Two-Dimensional Field Plots for an presentation of the same data in finite element format.

Three-Dimensional Field Plots

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
...
```

Finite Element Data

The zone header for a finite element zones lists the zone type, along with the number of nodes, elements and faces included in the zone. The following zone types are available for finite element data:

FELINESEG

FE line segments zones contain one-dimensional finite element zones. For the line segment element

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

FETRIANGLE

FE triangular zones contain two-dimensional finite elements defined by three nodes. For the triangle element type, each line of the connectivity list contains three node numbers that define a triangular element.

FEQUADRILATERAL

FE quadrilateral zones contain two-dimensional elements defined by four nodes. For the quadrilateral element type, each line of the connectivity list contains four node numbers that define a quadrilateral element.

If you need to mix quadrilateral and triangle elements, either use the polygonal zone type or use the quadrilateral element type with node numbers repeated to form triangles.

FEPOLYGON

FE polygonal zones contain two-dimensional elements defined by a varying number of nodes (three or greater). Note that Tecplot Focus cannot load this type of zone.

FETETRAHEDRON

FE tetrahedral zones contain three-dimensional elements defined by four nodes.

FEBRICK

FE brick zones contain three-dimensional elements defined by eight nodes. Tecplot 360 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 (where N# is the node number and M is the element number). 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 23 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 $N5_M$=$N6_M$=$N7_M$=$N8_M$.

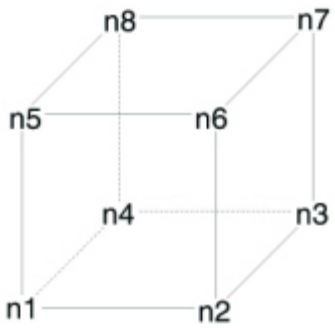


Figure 23. Basic brick connectivity.

FEPOLYHEDRAL

FE polyhedral zones contain elements with a varying number of faces. Each element has at least four faces. The faces are defined by any number of nodes (with a minimum of three nodes in each face). Note that Tecplot Focus cannot load this type of zone.

Refer to Zone Record for a complete list of the tokens included in the zone header.

After the zone header, the nodal data is listed. The nodal data contains the value of each variable for each node or element. Refer to Data for details on arranging the data. The information following the nodal data is dependent upon the zone type.

For cell-based finite element zone types (`FETRIANGLE`, `FEQUADRILATERAL`, `FETETRAHEDRON`, and `FEBRICK`), the nodal data is followed by the connectivity section. The connectivity section describes arrangement of cells, relative to one another. 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, `ZONETYPE=FETRIANGLE` has three numbers per line in the connectivity list. If nodes five, seven, and eight are connected, one line reads: `5 7 8`. Refer to Connectivity for details. You may also define Face Neighbors following the connectivity list. Face Neighbor Connections List for details.

For face-based zone type (`FEPOLYGON` and `FEPOLYHEDRAL`), the data section Data is followed by the zone footer and facemap data sections. Refer to Facemap Data for details.

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. NOTE: Connectivity and/or face neighbors cannot be shared when the face neighbor mode is set to Global.

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

X	Y	P_1	P_2	P_3
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 24.

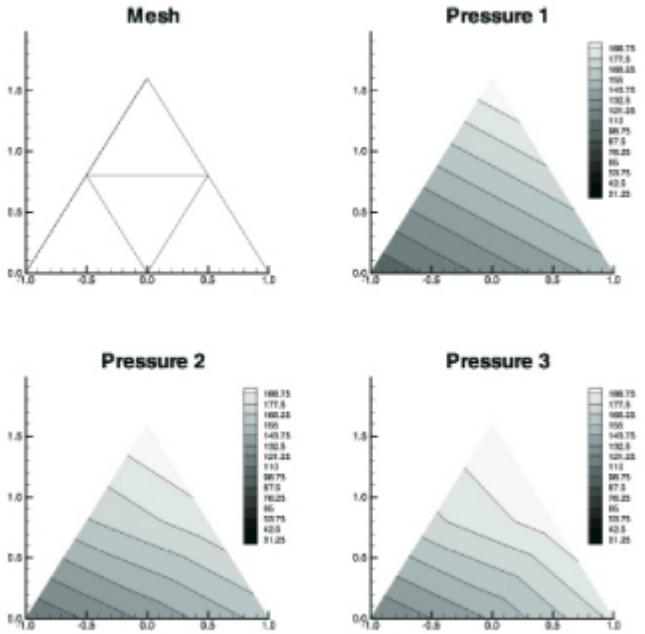


Figure 24. A plot of finite element zones.

```
TITLE = "Example: Variable and Connectivity List Sharing"
VARIABLES = "X", "Y", "P"
ZONE T="P_1", DATAPACKING=POINT, NODES=6, ELEMENTS=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, NODES=6, ELEMENTS=4, ZONETYPE=FETRIANGLE, VARSHARELIST =
([1, 2]=1), CONNECTIVITYSHAREZONE = 1
110 135 160 165 185 200

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

Finite Element Data Set Examples

Creating a finite element data set is generally more complicated than creating a similar-sized ordered data set. In addition to specifying all the data points, you must also specify the connectivity list. Consider the data shown in Table 4.

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 4. finite element Data

You can create a `POINT` Tecplot 360 data file for this data set as follows (a 2D mesh plot of this data set is shown in Figure 25:

```
TITLE = "Example: 2D Finite Element Data"
VARIABLES = "X", "Y", "P", "T"
ZONE NODES=8, ELEMENTS=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
0.0 0.0 50.0 1.0
1.0 0.0 100.0 1.4
3.0 0.0 200.0 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
```

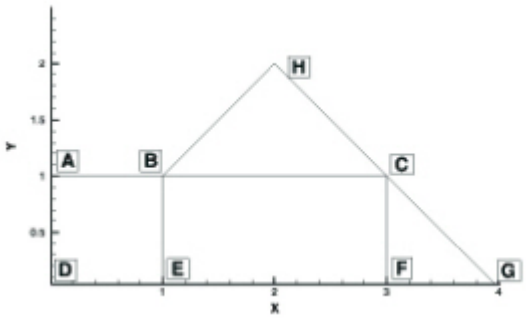


Figure 25. A mesh plot of 2D finite element data.

The `ZONE` record describes completely the form and format of the data set: there are eight nodes, indicated by the parameter `NODES=8`; four elements, indicated by the parameter `ELEMENTS=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 NODES=8, ELEMENTS=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 26. (You must also change the `ELEMENTS` parameter in the zone control line to specify five elements.)

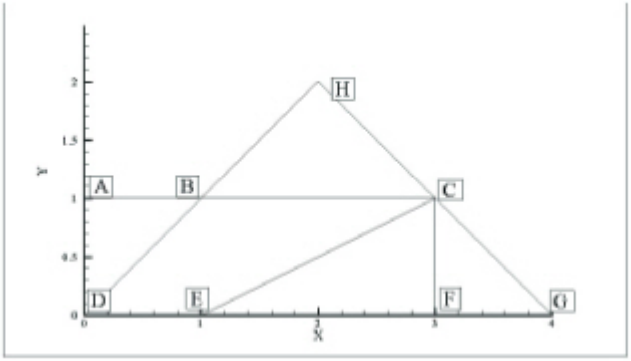


Figure 26. Finite element data of [#FIG:mesh-plot-3d-fe-data] with a different connectivity list

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

Two-Dimensional Field Plots

A 2D finite element data file is shown below:.

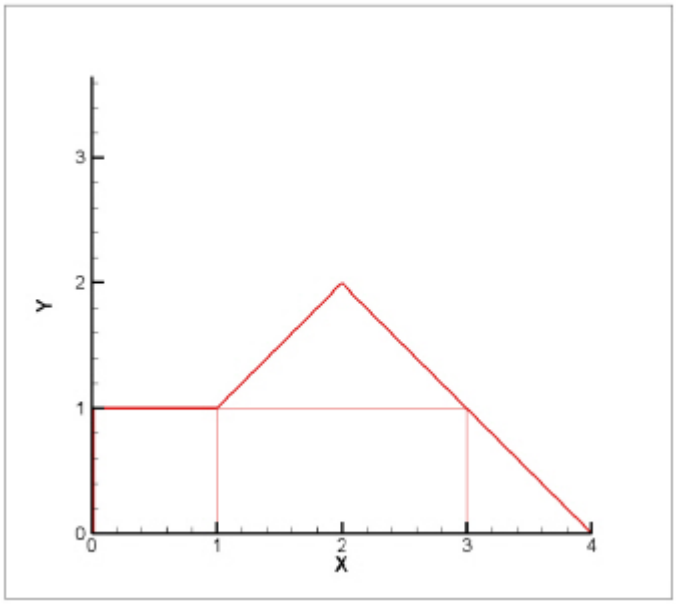


Figure 27. A 2D field plot.

```
TITLE = "Example: 2D Finite Element Data"
VARIABLES = "X", "Y", "P", "T"
ZONE NODES=8, ELEMENTS=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 eight rows of the zone) and four elements (the last four rows of the zone). Each row in the node matrix represents a given node. Each column in the row matrix corresponds to the value of each variable 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 one, 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 one, two, five and four.

Please refer to the User's Manual for information on ordered and FE data sets.

Triangle Data in BLOCK Format Example

An example of triangle element type finite element data is listed below. There are two variables (X, Y) and five data points. This data set is plotted in Figure 28 Each data point is labeled with its node number.

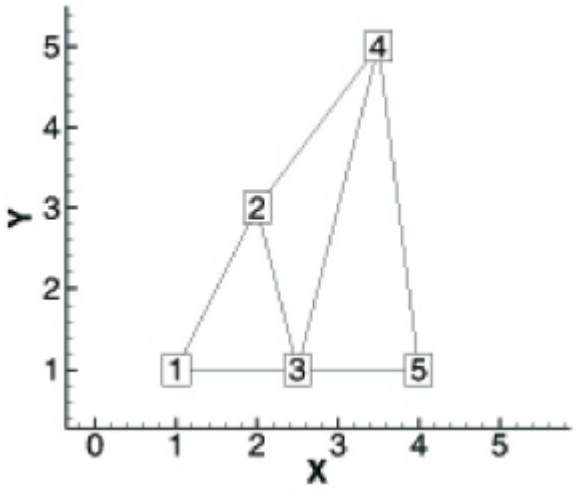


Figure 28. A finite element triangle data set.

In this example, each column of the data section corresponds to a node and each row to a variable. Each row of the connectivity list corresponds to a triangular element and each column specifies a node number.

```
VARIABLES = "X", "Y"
ZONE NODES=5, ELEMENTS=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

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

```
  INTEGER VAR
  .
  .
  WRITE (*,*) ´ZONE DATAPACKING=BLOCK, ZONETYPE=FETRIANGLE,NODES=´,NNODES,
',ELEMENTS=´,NELEM
  DO 1 VAR=1,NUMVAR
  DO 1 NODES=1,NNODES
    WRITE(*,*) VARRAY(VAR,NODES)
1  CONTINUE
  DO 2 M=1,NELEM
  DO 2 L=1,3
       WRITE (*,*) NDCNCT(M,L)
2  CONTINUE
```

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 360, although the connectivity list has changed to reflect the values of the 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" NODES=6, ELEMENTS=5,DATAPACKING=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" NODES=6, NV = 3, ELEMENTS=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
```

FE surface data

Finite element surface data specify node locations in three dimensions. Consider the data in Table 5. 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 elements are triangles and some are quadrilaterals. All the elements could be organized into one zone of element type Quadrilateral. However, 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

X	Y	Z
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 5. Data set with eleven nodes and three variables.

A Tecplot 360 data file for the data in Table 5 is shown below in `POINT` format and plotted in Figure 29:

```
TITLE = "Example: 3D FE-SURFACE ZONES"
VARIABLES = "X", "Y", "Z"
ZONE T="TRIANGLES", NODES=5, ELEMENTS=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", NODES=8, ELEMENTS=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", NODES=6, ELEMENTS=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
```

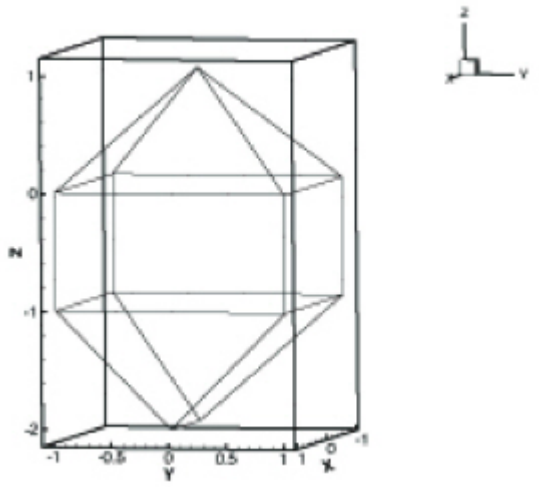


Figure 29. Three-dimensional mesh plot of finite element surface data.

FE Volume Data Files

Finite element volume data in Tecplot 360 is constructed from either tetrahedra 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.

Finite Element Volume - Brick Data Set

As a simple example of finite element volume brick data, consider the data in Table 6. 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

X	Y	Z	Temperature
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 6. Finite Element Volume - Brick Data Set. Data with 14 nodes and four variables.

In each element's connectivity list, Tecplot 360 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 23 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 duplicate nodes to create pyramids and wedges. Tecplot 360 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 NODES=14, ELEMENTS=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
1.0 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 NODES=14, ELEMENTS=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 30 shows the resulting mesh plot from the data set listed in this section.

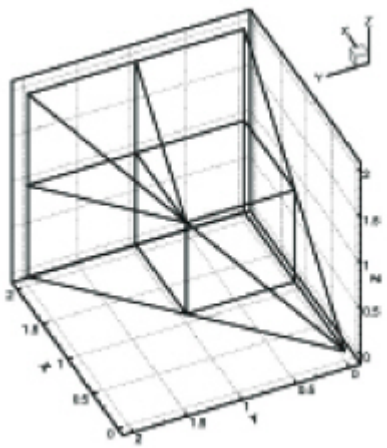


Figure 30. A finite element brick zone.

Finite Element Volume - Tetrahedral Data Set

As a simple example of a finite element volume data set using tetrahedral elements, consider the data in Table 7. 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
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 7. Finite Element Volume - Tetrahedral data set with 13 nodes and seven variables.

The data file in `POINT` format for the data in Table 7 is shown below, and plotted in Figure 31:

```
TITLE = "Example: FE-Volume Tetrahedral Data"
VARIABLES = "X", "Y", "Z", "C", "U", "V", "W"
ZONE NODES=13, ELEMENTS=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
```

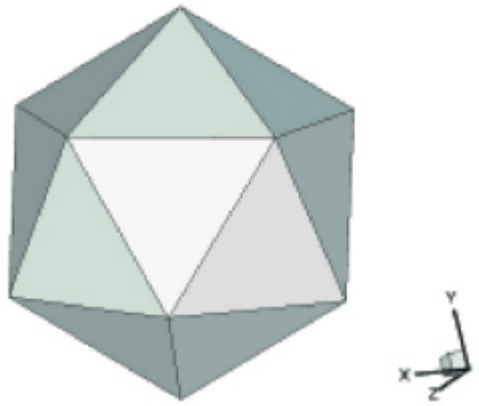
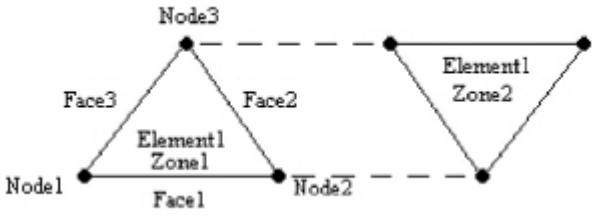


Figure 31. Finite element volume tetrahedral data.

Polygonal - simple example

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

A polygonal element in one zone connected to an element in another zone.




```
Zone ZoneType=FEPolygon Nodes=3 Faces=3 Elements=1 NumConnectedBoundaryFaces=2
TotalNumBoundaryConnections=1
```

...variable values in block format...

```
#face nodes
1 2
2 3
3 1
#left elements
1 1 1
#right elements (negative indicates boundary connections)
0 -1 0
#boundary connection counts
1
#boundary connection elements
1
#boundary connection zones
2
```

Polyhedral - complex example

Remember that Tecplot Focus cannot load face-based (polygonal or polyhedral) data.

A single tetrahedron bounded on face two by zone two (elements 13 and 14) and on face three by zone three (element 11).

```
Zone ZoneType=FEPolyhedron Nodes=4 Faces=4 Elements=1 TotalNumFaceNodes=12
NumConnectedBoundaryFaces=2 TotalNumBoundaryConnections=3
```

...variable values in block format...

```
#node count per face
3 3 3 3
#face nodes
1 2 3
1 4 2
2 4 3
3 4 1
#left elements (negative indicates boundary connection)
0 -1 -2 0
#right elements
1 1 1 1
#boundary connection counts
2 1
#boundary connection elements
13 14 11
```

```
#boundary connection zones
2 2 3
```

ASCII Data File Conversion to Binary

Although Tecplot 360 can read and write ASCII or binary data files, binary data files are more compact and are read into Tecplot 360 much more quickly. Even if you write ASCII files, for best performance, we recommend that you convert your data files to binary as a post-processing step, before interactive visualization.

In the `bin` folder of your Tecplot 360 distribution you will find Preplot, which converts ASCII to binary data files. You can also use Preplot to debug ASCII data files that Tecplot 360 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 on Windows platforms:

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 the standard set of Preplot options. 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 the standard set of preplot options, see the User's Manual.

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 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 three, Preplot outputs data points with I-index starting at six and ending at 34, skipping every other one, and J-index starting at one and ending at 21. For zone four, 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 one, and ending with the maximum index. The overall effect is to reduce the number of data points by a factor of about eight.

Reading SZL Data Files

In addition to writing `.plt` and `.szplt` files, TecIO includes an API that allows your application to read `.szplt` files, Tecplot's new high-performance subzone format. As we make improvements to the file format, we will update the TecIO library accordingly. The TecIO library does not support reading Tecplot ASCII `(.dat)` or binary `(.plt)` files.

TecIO's SZL reading capability is provided solely to provide compatibility with this new Tecplot file format. The performance improvement achieved by strategically reading partial zones (or subzones) is considered proprietary to Tecplot products and is not available to third-party developers.

As the SZL format evolves, we will strive to make each new version of the TecIO library a drop-in replacement for the previous version, so that you need only distribute the new version of the library to your users, or at most recompile your application. However, this may not always be feasible. We currently consider it unlikely that we will need to significantly change the SZL reading API in the future, but it is not impossible.

Tecplot is interested in your experiences with this API. Please contact support@tecplot.com with your questions or feedback.

Getting Started with the API

All the functions you use to read SZL files are declared in `TECIO.h` (C or C++ programmers) or in `tecio.for` or `tecio.f90` (FORTRAN programmers) along with the other TecIO functions. The names of the functions start with `tec` (lower-case) and contain `read` or `get`, for example `tecZonePolyGetFaceNodes`. In the C header file, they follow a comment reading *SZL file reading routines. The header files are the authoritative reference to the API.*

There are more than a hundred individual functions in the SZL reading API, but don't let that intimidate you. Nearly all map to specific Tecplot data objects or to specific fields within those objects. All of the functions in the API use simple data types, or arrays of them. You can easily obtain only the information your application is interested in by simply not calling API functions for data you don't need. Furthermore, if you are interested only in the data in the file and do not want to read, say, geometries or text objects, you will not ever need to call a substantial number of functions.

For example, there is a group of four functions related to getting information about an arrowhead attached to a line geometry: one to get its angle, one to get its attachment (i.e., which ends of the line segment have an arrowhead), one to get its size, and one to get its style. These functions all have names beginning with `tecGeomArrowhead`, so it is easy to find them when you want to obtain arrowhead details. But if you are not interested in arrowheads, you can ignore them entirely.

You may want to develop an object-oriented layer on top of the TecIO API to better interface with your application. Since each application's needs are different, we leave this to you. If you come up with something generally useful, we encourage you to make it available to others.

Passing Arguments and Receiving Values

Nearly all SZL reading functions pass one or more values back to the caller through output arguments in each function's argument list rather than using return values. The output arguments are pointers or references to the caller's variable in which the returned data should be stored. Each function in the API gets one data item from the file, so in most cases there is only one output parameter. An exception to this rule is made for sizes of shapes (which are passed back as separate length and width values from functions such as `tecGeomRectangleGetSize`) and coordinates (which are passed back as separate X, Y, and Z values from, for example, `tecTextGetAnchorPos`). If you need one of these related values, you will almost always need all of them, and you shouldn't need to call two or three separate function calls to do the job!

Many integer values in the API represent not numbers but rather enumerations specific to Tecplot products. For example, `tecGeomGetLinePattern` returns an integer representing the line pattern, where each value has a specific meaning. The values for such fields are the same as those used in the classic binary file writing API described in Binary Data and are documented in Meaningful Integer Values.

Indexing and Zero Values

All indexing in Tecplot data files (including zones and variables) is 1-based, so the last index is exactly equal to (not one less than) the count of the element type. An exception is nodes in finite-element zones, which can be specified to be zero-based. C and C++ programmers in particular should be wary of off-by-one errors when dealing with Tecplot data.

Zero is used in some functions to indicate the *lack* of a value; for example, if a geometry has no zone, `tecGeomGetZone` passes back `0` for that zone.

Errors

With only two exceptions, return values in the API are used exclusively to indicate success or failure. Functions return zero on success and non-zero on failure.

Errors are often caused by asking for a data element that does not exist (for example, zone 10 in a file that contains only nine zones). This may be avoided by using the provided `GetNum` functions (such as `tecDataSetGetNumZones`) to determine how many items (in this case, zones) exist before attempting to retrieve them.

Another potential error is calling a function to get attributes of a geometry object when that geometry is not of the proper type (for example, trying to get the radius of a square). Be sure to check the geometry's type before getting its attributes.

Errors may also indicate a corrupted or truncated file due to a file transfer error, insufficient available memory, or a programming error such as an invalid pointer.

Once a file has been opened successfully, and reading is underway, it is generally best to advise users of an error condition, but offer to continue trying to read additional items from the file if this is at all possible.

API Overview

The 100+ functions of the TecIO SZL reading API are divided into the following broad categories. (Some functions are shown here in more than one category to make it easier to find the one you need.) In some cases, we have divided larger categories into subcategories as well.

- Files and Data Sets
- Geometries
- Text

- Custom Label Sets
- Strings
- Auxiliary Data
- Variables
- Zones
- Connectivity
- Data Values

Functions are generally listed alphabetically, except in cases where you are likely to need a particular function first. For example, when reading text objects from a file, you will need to know how many there are before you can read any of them, so `tecTextGetNumTexts` is listed first.

Files and Data Sets

These functions open and close files and provide information about the data sets in open files. See Opening and Closing Files for details.

Function	Input	Output
tecFileReaderOpen	Filename	File Handle
tecFileReaderClose	File Handle	None
tecFileGetType	File Handle	File Type
tecDataSetGetTitle	File Handle	Title
tecDataSetGetNumVars	File Handle	Count of Variables in Data Set
tecDataSetGetNumZones	File Handle	Count of Zones in Data Set
tecDataSetAuxDataGetNumItems	File Handle	Count of Data Set Auxiliary Data Items
tecDataSetAuxDataGetItem	File Handle, Auxiliary Data Item Index	Name and Value of specified Data Set Auxiliary Data Item

Geometries

These functions read geometries (that is, shapes) from the data file. See Reading Geometry and Text for details.

The following functions are used with all types of geometries.

Function	Input	Output
tecGeomGetNumGeoms	File Handle	Count of Geometries in Data Set

Function	Input	Output
tecGeomGetType	File Handle, Geometry Index	Geometry Type
tecGeomGetAnchorPos	File Handle, Geometry Index	X, Y, and Z coordinates
tecGeomGetClipping	File Handle, Geometry Index	Clipping
tecGeomGetColor	File Handle, Geometry Index	Color
tecGeomGetCoordMode	File Handle, Geometry Index	Coordinate Mode
tecGeomGetFillColor	File Handle, Geometry Index	Color
tecGeomGetLinePattern	File Handle, Geometry Index	Line Pattern
tecGeomGetLineThickness	File Handle, Geometry Index	Thickness
tecGeomGetMacroFunctionCmd	File Handle, Geometry Index	Name of Attached Macro Function
tecGeomGetPatternLength	File Handle, Geometry Index	Pattern Length
tecGeomGetScope	File Handle, Geometry Index	Scope
tecGeomGetZone	File Handle, Geometry Index	Zone Index
tecGeomIsAttached	File Handle, Geometry Index	Boolean
tecGeomIsFilled	File Handle, Geometry Index	Boolean

Line Segments

The following functions are used with line segments (polylines) and the arrowheads optionally attached to them.

Function	Input	Output
tecGeomLineGetSegmentCount	File Handle, Geometry Index	Count of Line Segments in Geometry
tecGeomLineSegmentGetPointCount	File Handle, Geometry Index, Line Segment Index	Count of Points in Line Segment
tecGeomLineGetPoint	File Handle, Geometry Index, Line Segment Index, Point Index	X, Y, and Z coordinates
tecGeomArrowheadGetAngle	File Handle, Geometry Index	Arrowhead Angle
tecGeomArrowheadGetAttach	File Handle, Geometry Index	Arrowhead Attachment
tecGeomArrowheadGetSize	File Handle, Geometry Index	Arrowhead Size
tecGeomArrowheadGetStyle	File Handle, Geometry Index	Arrowhead Style

Other Shapes

The following functions are used with other shapes.

Function	Input	Output
tecGeomRectangleGetSize	File Handle, Geometry Index	Width and Height
tecGeomSquareGetSize	File Handle, Geometry Index	Size (i.e., Width/Height are same)
tecGeomCircleGetRadius	File Handle, Geometry Index	Radius
tecGeomEllipseGetNumPoints	File Handle, Geometry Index	Count of Points in Ellipse
tecGeomEllipseGetSize	File Handle, Geometry Index	Width and Height

Text

The following functions read text objects and their attributes from a data set. See Reading Geometry and Text for details.

Function	Input	Output
tecTextGetNumTexts	File Handle	Count of Text Objects
tecTextGetAnchor	File Handle, Text Index	Text Anchor
tecTextGetAnchorPos	File Handle, Text Index	X, Y, and Z Coordinates
tecTextGetAngle	File Handle, Text Index	Angle
tecTextGetClipping	File Handle, Text Index	Clipping
tecTextGetColor	File Handle, Text Index	Color
tecTextGetCoordMode	File Handle, Text Index	Coordinate Mode
tecTextGetHeight	File Handle, Text Index	Height
tecTextGetLineSpacing	File Handle, Text Index	Line Spacing
tecTextGetMacroFunctionCmd	File Handle, Text Index	Name of Attached Macro Function
tecTextGetScope	File Handle, Text Index	Scope
tecTextGetSizeUnits	File Handle, Text Index	Text Size Units
tecTextGetString	File Handle, Text Index	Text String
tecTextGetTypeface	File Handle, Text Index	Typeface/Font
tecTextGetZone	File Handle, Text Index	Zone Index
tecTextIsAttached	File Handle, Text Index	Boolean
tecTextIsBold	File Handle, Text Index	Boolean
tecTextIsItalic	File Handle, Text Index	Boolean

Text Box

The following functions read attributes of text boxes, which provide backgrounds and borders around text objects. Each text box is associated with one text item.

Function	Input	Output
tecTextBoxGetColor	File Handle, Text Index	Color
tecTextBoxGetFillColor	File Handle, Text Index	Color
tecTextBoxGetLineThickness	File Handle, Text Index	Line Thickness
tecTextBoxGetMargin	File Handle, Text Index	Margin
tecTextBoxGetType	File Handle, Text Index	Text Box Type

Custom Label Sets

The following functions read custom labels, which can be used for axis labels, legend text, and tick mark labels. Each custom label set is a string containing quoted label items separated by commas or spaces. See Reading Custom Labels for details.

Function	Input	Output
tecCustomLabelsGetNumSets	File Handle	Count of Custom Label Sets
tecCustomLabelsGetSet	File Handle, Custom Label Set Index	Custom Label Set (A single string containing quoted labels separated by spaces or commas)

Strings

Strings are UTF-8 encoded and null-terminated. When a string is passed back to your application by the a SZL reader function, the memory in which it resides is owned by the TecIO library and should be freed when you are done with it. See Dealing with Strings for details.

Function	Input	Output
tecStringLength	Pointer to String	Length it Bytes
tecStringFree	Pointer to String	None

Auxiliary Data

Auxiliary data records can be attached to three objects in a Tecplot file: the data set itself, a zone, and a variable. Each record is a name-value pair. See Reading Auxiliary Data for details.

The following functions are used to read auxiliary data items.

Function	Input	Output
tecDataSetAuxDataGetNumItems	File Handle	Count of Data Set Auxiliary Data Items
tecDataSetAuxDataGetItem	File Handle, Auxiliary Data Item Index	Name and Value of specified Data Set Auxiliary Data Item
tecVarAuxDataGetNumItems	File Handle, Variable Index	Count of Variable Auxiliary Data Items
tecVarAuxDataGetItem	File Handle, Variable Index, Auxiliary Data Item Index	Name and Value of specified Variable Auxiliary Data Item
tecZoneAuxDataGetNumItems	File Handle, Zone Index	Count of Zone Auxiliary Data Items
tecZoneAuxDataGetItem	File Handle, Zone Index, Auxiliary Data Item Index	Name and Value of specified Zone Auxiliary Data Item

Variables

Variables are named collections of values with a data type and value location. The following functions retrieve information about variables. See Reading Data for details.

Function	Input	Output
tecDataSetGetNumVars	File Handle	Count of Variables in Data Set
tecVarGetName	File Handle, Variable Index	Variable Name
tecVarIsEnabled	File Handle, Variable Index	Boolean
tecZoneVarGetType	File Handle, Zone Index, Variable Index	Data Type
tecZoneVarGetValueLocation	File Handle, Zone Index, Variable Index	Value Location
tecZoneVarIsPassive	File Handle, Zone Index, Variable Index	Boolean
tecZoneVarGetSharedZone	File Handle, Zone Index, Variable Index	Shared Zone Index (0 if none)
tecVarAuxDataGetNumItems	File Handle, Variable Index	Count of Variable Auxiliary Data Items
tecVarAuxDataGetItem	File Handle, Variable Index, Auxiliary Data Item Index	Name and Value of specified Variable Auxiliary Data Item

Zones

Zones are used to organize data spatially, chronologically, or by some other characteristic. These functions get information about zones. See Reading Data for details.

Function	Input	Output
tecDataSetGetNumZones	File Handle	Count of Zones in Data Set
tecZoneGetTitle	File Handle, Zone Index	Zone Title
tecZoneGetType	File Handle, Zone Index	Zone Type
tecZoneIsEnabled	File Handle, Zone Index	Boolean
tecZoneGetIJK	File Handle, Zone Index	I, J, K extent of Specified Zone for ordered zoness number of points and elements for FE zones (see Reading Data)
tecZoneGetNumSections	File Handle, Zone Index	Number of sections in Specified Zone for FE zones of ZoneType_FEMixed (see Reading Data)
tecZoneGetSectionMetrics	File Handle, Zone Index, Section Index	CellShape, GridOrder, BasisFunction, NumElemsInSection, and NunNodesPerCell for Secified Section of Specified FEMixed Zone (see Reading Data)
tecZoneGetSolutionTime	File Handle, Zone Index	Solution Time
tecZoneGetStrandID	File Handle, Zone Index	Strand ID
tecZoneVarGetType	File Handle, Zone Index, Variable Index	Data Type
tecZoneVarGetValueLocation	File Handle, Zone Index, Variable Index	Value Location
tecZoneVarIsPassive	File Handle, Zone Index, Variable Index	Boolean
tecZoneVarGetSharedZone	File Handle, Zone Index, Variable Index	Shared Zone Index (0 if none)
tecZoneConnectivityGetSharedZone	File Handle, Zone Index, Variable Index	Shared Zone Index (0 if none)
tecZoneAuxDataGetNumItems	File Handle, Zone Index	Count of Zone Auxiliary Data Items

Function	Input	Output
tecZoneAuxDataGetItem	File Handle, Zone Index, Auxiliary Data Item Index	Name and Value of specified Zone Auxiliary Data Item

Connectivity

Connectivity refers to how the nodes and cells in finite element zones are connected and encompasses the node map and face neighbors. See Reading Connectivity Data for details.

Node Map

The following functions are used to read the node map from finite element zones. The node map indicates which nodes are part of which elements.

Function	Input	Output
tecZoneNodeMapIs64Bit	File Handle, Zone Index	Boolean
tecZoneNodeMapGetNumValues	File Handle, Zone Index, Number of Cells	Count of Node Map Values for the specified Number of Cells
tecZoneNodeMapGet	File Handle, Zone Index	Array of Node Map Values
tecZoneNodeMapGet64	File Handle, Zone Index	Array of Node Map Values
tecZoneSectionNodeMapGetNumValues	File Handle, Zone Index, Section Index	Count of Node Map Values for the specified Section and Number of Cells

Face Neighbors

The following functions read face neighbor data, which indicates which elements are adjacent.

Function	Input	Output
tecZoneFaceNbrGetMode	File Handle, Zone Index	Face Neighbor Mode
tecZoneFaceNbrGetNumValues	File Handle, Zone Index	Count of Face Neighbor Values
tecZoneFaceNbrsAre64Bit	File Handle, Zone Index	Boolean
tecZoneFaceNbrGetConnections	File Handle, Zone Index	Array of Face Neighbor Connections
tecZoneFaceNbrGetConnections64	File Handle, Zone Index	Array of Face Neighbor Connections

The connectivity functions below are not currently implemented because they apply to face-based (polygonal/polyhedral) zones, which SZL files cannot currently contain. They are provided so that the interface does not need to be changed when this capability is implemented.

- tecZonePolyGetBoundaryConnectionCounts
- tecZonePolyGetBoundaryConnections
- tecZonePolyGetFaceElems
- tecZonePolyGetFaceNodeCounts
- tecZonePolyGetFaceNodes
- tecZonePolyGetNumConnectedBoundaryFaces
- tecZonePolyGetTotalNumFaceNodes
- tecZonePolyGetTotalNumBoundaryConnections

Data Values

SZL files can contain IJK indexed data and classic finite-element (FE) volume data. The following functions are used to read data values from all types of zones. See Accessing Data Values for more details.

Function	Input	Output
tecZoneVarGetNumValues	File Handle, Zone Index, Variable Index	Count of Values in Variable for Specified Zone
tecZoneVarGetDoubleValues	File Handle, Zone Index, Variable Index, Start Index, Number of Values	Array of Data Values
tecZoneVarGetFloatValues	File Handle, Zone Index, Variable Index, Start Index, Number of Values	Array of Data Values
tecZoneVarGetInt16Values	File Handle, Zone Index, Variable Index, Start Index, Number of Values	Array of Data Values
tecZoneVarGetInt32Values	File Handle, Zone Index, Variable Index, Start Index, Number of Values	Array of Data Values
tecZoneVarGetUint8Values	File Handle, Zone Index, Variable Index, Start Index, Number of Values	Array of Data Values

Opening and Closing Files

The functions tecFileReaderOpen and tecFileReaderClose are used to open and close files. tecFileReaderOpen passes back a file handle—a pointer to a data structure internal to the library that you should treat as opaque. This handle is a required parameter to virtually all other functions in the

SZL reading API.

It is possible to open more than one file at a time, passing the handle for the specific file you want to read when requesting data elements from it. The number of simultaneously open files is limited only by memory and/or the underlying operating system.

You should close files when you are done reading them. However, it is fine to leave a file open for a long time (for example, while a user works with the data in it in an interactive fashion). If your program is terminating for whatever reason, it is not strictly necessary to close open SZL files; the operating system will do it for you. It is not possible to lose data in this scenario, as the SZL reading API does not write to any files.

Unlike the close function in many APIs, `tecFileReaderClose` requires that you pass in a pointer to the variable where you originally stored the file handle when you called `tecFileReaderOpen`, *not the file handle itself*. TecIO will store a null pointer in this variable to help make sure you do not try to use the file handle again after closing the file. (This is necessary because file handles are guaranteed unique only as long as the file is open. Two files open at different times might end up with the same handle.)

After opening the file, you can get useful basic information about the dataset within it using the following informational functions:

- `tecDataSetGetTitle` - title of data set
- `tecDataSetGetNumVars` - number of variables
- `tecDataSetGetNumZones` - number of zones
- `tecFileGetType` - whether the file contains a grid, a solution, or both

Dealing with Strings

Strings (for example, zone names, variable names, auxiliary data, and the text contained by text objects) are C-style zero-terminated character (i.e., unsigned byte) arrays using UTF-8 encoding. The string itself is held in memory that belongs to the TecIO library, and you receive a pointer to it. You should not modify this memory, but it is suitable for read-only uses such as printing. If you wish to manipulate the string or keep your own copy of it, you should copy it to memory allocated and owned by your application.

The function `tecStringLength` is provided to obtain the length of a string. You should use this in place of e.g. `strlen` since it checks the validity of the pointer before getting the length. It returns -1 if the pointer is not valid (that is, if it is null). Note that this returns the length of the string in bytes, not in characters; some characters in the UTF-8 encoding require multiple bytes. Decode the string to an internal Unicode representation if you need to know how many characters (code points) it contains.

Call `tecStringFree` when you are done with the string to inform the TecIO library that you no longer need access to it. Failure to do so may result in unneecssary memory usage, as the memory used for a zone or variable cannot be reused while any references to its contents are outstanding.

The following C++ code fragment illustrates how to obtain the title of a dataset.

```
   char* dataSetTitle = NULL;
   err = tecDataSetGetTitle(fileHandle, &dataSetTitle);
   /* Do something with dataSetTitle */
   tecStringFree(&dataSetTitle);
```

Below is an equivalent fragment in FORTRAN.

```
   type(c_ptr) :: fileHandle = C_NULL_PTR
   type(c_ptr) :: stringCPtr = C_NULL_PTR
   character(256) dataSetTitle, zoneTitle
   INTEGER(4) zone

   i = tecDataSetGetTitle(fileHandle, stringCPtr)
   call copyCharArrayToString(stringCPtr, tecStringLength(stringCPtr), dataSetTitle)
! Do something with dataSetTitle ...
   call tecStringFree(stringCPtr)
```

There are no functions in the API that accept strings as inputs aside from `tecStringFree`.

Reading Data

SZL files can contain classic (cell-based) finite element and IJK-ordered zones. Currently, SZL files cannot contain face-based polygonal or polyhedral data[1]. Before reading data, you will want to find out the following:

- The number of variables. API: `tecDataSetGetNumVars`
- The number of zones. API: `tecDataSetGetNumZones`
- The indexes of the zone(s) and variable(s) you are interested in, if you do not want all variables. There are no API functions to locate a zone or variable by name; instead, iterate over all zones and variables, get the name of each, see if it's one you want, and remember its index (e.g. in a vector) if so. API: `tecZoneGetTitle`, `tecVarGetName`

1. The TecIO SZL reading API includes stubs for the functions necessary to read face-based zones, so that if and when the feature is supported in the future, we do not need to change the API. However, you will find no reason to call them at this point.

For each zone and variable to be read, you will want to find out:

- The dimensions of the zone. You use one function, `tecZoneGetIJK`, to retrieve this information regardless of the zone type. For IJK-ordered zones, this call provides the maximum value of the I, J, and K indices (some of these will be 1 if the zone has fewer than three dimensions). For classic (cell-based) FE zones, it provides the number of points as I and the number of elements as J; K is not used and will receive 0.

- Information about the zone: type, solution time, parent zone, time strand, and whether it is enabled or disabled. If your zone is of type FEMixed it may be composed of multiple sections with different cell shapes, grid-orders (>1 is a higher-order element), basis functions and number of nodes per cell. Your application may not need all of this information, so you may skip some of these calls. API: `tecZoneGetType`, `tecZoneGetSolutionTime`, `tecZoneGetStrandID`, `tecZoneIsEnabled`, `tecZoneNumSections`, `tecZoneGetSectionMetrics`.
- Information about the variable: whether it is enabled or not, its type, value location (nodal or cell-centered), whether it is shared, and whether it is passive. Much of this information is with respect to a specific zone. Again, you may skip any calls for information your application does not need. API: `tecVarIsEnabled`, `tecZoneVarGetType`, `tecZoneVarGetValueLocation`, `tecZoneVarGetSharedZone`, `tecZoneVarIsPassive`.

  Note that the variable may be shared from another zone, in which case you should prepare to read the variable from that zone if it is not already in memory, or copy or store a reference to its data if it has already been loaded.

  Variables that are not used in a particular zone may be marked as passive. In this case, no data is stored in the file for the variable, and zero will be returned for every location.

- The number of values. This will tell you how many elements you need to allocate to read the entire variable into memory. (The type of the variable implies how many bytes you need per element.) API: `tecZoneVarGetNumValues`

After determining the dimensions and type, you can allocate memory for the data and read the values using one of the `tecZoneVarGetValues` functions: `tecZoneVarGetDoubleValues`, `tecZoneVarGetFloatValues`, `tecZoneVarGetInt16Values`, `tecZoneVarGetInt32Values`, or `tecZoneVarGetUint8Values`, depending on the type of the variable.

The TecIO library does not require that you read all the values in a zone's variable at once—or for that matter, at all. All of the `tecZoneVarGetValues` functions include two parameters, `startIndex` and `numValues`, that let you indicate where to start and how many values you want to receive. (Like all indexes in the TecIO library, `startIndex` is 1-based: the first element is index 1.) You may read a zone in chunks by varying these parameters from call to call.

Even though the actual zone may have 1, 2, or 3 dimensions, the values array is one-dimensional. It is up to your application to find the index of the desired node or cell. More on this in Accessing Data Values.

Reading Connectivity Data

Classic FE (cell-based) zones also include node map and face neighbor information (collectively called the zone's *connectivity*) to tell you which nodes belong to which cell and which cells are adjacent. Cells are considered adjacent (connected, or neighbors) if one of their faces' nodes consist of exactly the same points as the nodes of a face of another cell. The *nodemap* is an array specifying the nodes that constitute each cell. *Face neighbors* are the faces that are shared between cells.

When loading data, Tecplot 360 or Tecplot Focus automatically detects which cells share faces based on the IJK indexing of the cells (for indexed zones) or the entries in the node map (for finite element zones). You should be prepared to do the same if you need this information. Additional face neighbors that cannot be detected in this way can be specified separately in the file, so you should also be prepared to read this data if needed.

Connectivity data is stored per zone and so does not need to be re-read for every variable in the zone. Like variables, connectivity can be, and often is, shared among zones. Call `tecZoneConnectivityGetSharedZone` to find out which zone holds the connectivity data for the zone of interest; this will be zero if the zone has its own connectivity information. If the zone uses another zone's connectivity data, you will want to read it from that zone if it is not already in memory, or use a pointer or a copy of the data if it has already been loaded. You will need to keep the node map on hand in order to find the nodes of each cell.

Most applications will need to read the nodemap. To do this:

- Call `tecZoneNodeMapIs64Bit` to find out if the node map entries are 64-bit or 32-bit.
- Call `tecZoneNodeMapGetNumValues` to find out how many entries are in the node map, or `tecZoneSectionNodeMapGetNumValues` to find out how many entries are in the node map for a Section of an FEMixed zone.
- Using the information from these two calls, allocate the memory to read the node map.
- Finally, call either `tecZoneNodeMapGet` or `tecZoneNodeMapGet64` depending on the result of `TecZoneNodeMapIs64Bit` to actually read the nodemap.

Your application may not need the face neighbor data. If you need it, the process is similar:

- Call `tecZoneFaceNbrsAre64Bit` to find out if the face neighbor entries are 64-bit or 32-bit.
- Call `tecZoneFaceNbrGetMode` to get the face neighbor mode for the zone. This will determine how many entries in the face neighbor array there are for each cell (see the output function `TECFACE142` for a discussion).
- Call `tecZoneFaceNbrGetNumValues` to get the total number of face neighbor entries.
- Use this information to calculate how much memory is needed to load the face neighbor data and allocate it.
- Finally, call `tecZoneFaceNbrGetConnections` or `tecZoneFaceNbrGetConnections64` to read the face neighbors.

Accessing Data Values

Accessing data from a variable in an IJK-ordered zone is straightforward. Although stored and delivered via a one-dimensional array, the values read can represent a 1, 2, or 3-dimensional array, with the I-index varying the fastest and the K index the slowest. It is possible to write a generic function to retrieve the desired value based on its I, J, and K indexes, since `tecZoneGetIJK` returns 1 for a dimension that is not used. For example, if I, J, and K are the (1-based) indexes of the desired value,

you can get the index of the element in the values array with the following formula:

C/C++ (zero-based): `I-1 + (J-1) * IMax + (K-1) * IMax * JMax`

FORTRAN (one-based): `I + (J-1) * IMax + (K-1) * IMax * JMax`

In one way, accessing data from a classic finite-element (cell-based) zone is more straightforward than accessing data in an ordered zone, but it is more complicated in other ways. The variable values read from the file are a one-dimensional array of the values corresponding to the cells (for cell-centered data) or the nodes (for nodal data) in the variable. So if you know the cell or node number, it is simple to obtain the corresponding data from a variable that is cell-centered or nodal, respectively.

If you have a cell number, but the data is nodal, you will need to first access the nodemap to see what nodes make up the desired cell, then get the data at those nodes, perhaps averaging it to a single value.

Conversely, if you have a node number, but the data is cell-centered, you need to determine what cells the node is a part of, which requires searching the nodemap for those cells and noting the indexes of the nodes in which it participates. If the node map needs to be searched repeatedly, you may want to create an inverse map and access the cells through that, as this will yield better performance.

Accessing FE data by its spatial coordinates will require finding the node or cell closest to the desired point based on the X, Y, and Z variables, which define the geometry of the zone.

Reading Auxiliary Data

Tecplot data files can store named values referred to as *auxiliary data*. Auxiliary data is often used to store metadata about a dataset or to hold the results of aggregate calculations such as overall lift or maximum and minimum pressure. In Tecplot data files, auxiliary data can be attached at three points: the dataset itself, the zone, and the variable. Each dataset, zone, or variable may have any number of auxiliary data records attached. Reading each type is straightforward and very similar:

- Obtain the number of items of the desired type using `tecDataSetAuxDataGetNumItems`, `tecZoneAuxDataGetNumItems`, or `tecVarAuxDataGetNumItems`.

  Zone and variable aux data is attached to specific zones or variables, and you must specify the index of the zone or variable of interest in the corresponding function calls.

- For each auxiliary data item, call `tecDataSetAuxDataGetItem`, or tecZoneAuxDataGetItem, tecVarAuxDataGetItem to get the item's name and value.

  Both name and value are strings; remember to free the strings when you are done with them (see Dealing with Strings).

As with zones and variables, you cannot access auxiliary data records by name. If you are looking for a particular record, you must check the name of each record of the desired type to see if it is the one you want. Records may be in any order, so it may be necessary to look at them all.

Auxiliary data values may represent numeric quantities or, for example, a comma-separated list of values or other record-style data. These are still returned as strings; you must convert or parse them yourself.

Reading Geometry and Text

Tecplot data files may contain line segments (actually, collections of linked line segments called *polylines*, though the API calls them just lines) circles, ellipses, squares, rectangles, and text objects. Collectively, they are referred to as *geometries*.

In most cases geometry has been added to aid in the understanding of the data by human beings; these shapes are not, strictly speaking, part of the data, and many applications can ignore them. But shapes can also be used to indicate areas of interest within a data set for further processing by software.

Tecplot data files may also contain text objects. Again, in most cases this is annotation for humans who will be viewing the data, but might also be used to hold further processing instructions that will be used by software. (In most cases it would be better to use auxiliary data for this, however.)

There are two sets of functions involved in reading geometries. One set applies to all kinds of geometries and are used to get general information about the geometry: its type, its scope, its coordinate mode, its fill color, its line characteristics (color, thickness, pattern), its anchor point, and so on. The other set applies to the special characteristics of specific types of geometries. For example, lines have segments and so there is `tecGeomLineGetSegmentCount` to get the number of segments, `tecGeomLineSegmentGetPointCount` to get the number of points in each segment, and `tecGeomLineGetPoint` to get each individual point.

Reading text is similar; the `tecText` functions are used to obtain characteristics of text objects, starting with the number of text objects (`tecTextGetNumTexts`) and proceeding to read the characteristics of specific text objects. While text objects have some things in common with geometries, they are sufficiently different that they are not classified as geometry objects inside Tecplot data files. If you are interested only in text objects, you do not have to iterate over all geometries to determine which are text objects.

Reading Custom Labels

Custom labels can be used for axis labels, legend text, and tick mark labels. Most applications will not need to read this data.

The first label in a label set 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. You can get the number of custom label sets using `tecCustomLabelsGetNumSets` and retrieve each set with `tecCustomLabelsGetSet`. Each set is passed back as a single string containing the labels enclosed in double quote marks. The quoted labels may be separated by spaces or commas.

Example Code

The `rewriteszl` example is a simple program that reads a SZL data file using the TecIO SZL reading API and writes it to a new SZL file using the new SZL writing API (see Writing SZL Data Files (New API)). The resulting file is functionally equivalent to the original, though it may not be identical at the binary level.

As such, the code illustrates reading every kind of data that can be stored in a SZL file. It also prints information about each data object as it is read, which may make it helpful as a diagnostic tool. This example is provided in both C++ `(.cpp)` and FORTRAN `(.F, .f90)` versions. Visual Studio solution files are also provided.

Writing SZL Data Files (New API)

Tecplot 360 and Tecplot Focus 2017 R2 introduce a new TecIO API for writing SZL `(.szplt)` data. This new API has a more flexible calling order, making it easier to write your data as you have it, instead of having to hold on to it and deliver it all at once to the library. Additionally, the new API brings:

- 64-bit indexing for support of zones with more than two billion nodes.
- Support for integer data types of various sizes in addition to floating-point types.
- The ability to open any number of files for writing simultaneously.

The new API does not currently support writing Tecplot `.plt` files, although we are considering adding this capability in the future. For now, `.plt` files can written using the classic API (see Binary Data), which will continue to be available indefinitely, and the classic API will also continue to be able to output `.szplt` files if your application already uses it for this purpose.

However, for new development, we encourage you to use the new API. This API is where Tecplot will be focusing TecIO development efforts going forward, and we plan several important improvements in upcoming releases.

Tecplot is keenly interested in your experiences with this API. Please contact support@tecplot.com with your questions or feedback.

Getting Started with the API

All the functions you use to write SZL files are declared in `TECIO.h` (C or C++ programmers) or in `tecio.for` or `tecio.f90` (FORTRAN programmers) along with the other TecIO functions. The names of the functions start with `tec` (lower-case) and mostly include `write, set, create,` or `add,` for example `tecDataSetAddAuxData`. In the C header file, they follow a comment reading *New TecIO output routines*. The header files are the authoritative reference to the API.

There are more than sixty individual functions in the new writing API, but about half have to do with geometries or text objects. Most applications will write only data and, potentially, auxiliary data, and

will not ever need to call a substantial number of functions.

Also, there are a handful of functions dealing with polytope (face-based) zones, which `.szplt` files cannot currently contain and which you will therefore not have occasion to call at this time. A solver that writes only a single type of zone might need just a dozen or so API functions.

Passing Arguments and Receiving Values

Scalar values (i.e., single integers) are passed by value. Other data types, such as strings and arrays of data elements, are passed by reference. (FORTRAN programmers need not worry about this detail.)

As this is a writing API, most functions do not need to pass any data back to the caller. For the rare function that gives data to the caller, an output argument (i.e., a pointer to memory owned by your application) is used. For example, `tecFileWriterOpen` passes back the file handle of the opened file using a pointer to a memory space you provide; the `tecZoneCreate` functions pass back the index of the new zone.

Many integer values in the API represent not numbers but rather enumerations specific to Tecplot products. For example, `tecGeomSetLineInfo` requires an integer representing the line pattern, where each value has a specific meaning. The values for such fields are the same as those used in the classic binary file writing API described in Binary Data and are documented in Meaningful Integer Values.

Indexing and Zero Values

By default, indexing in Tecplot data files (including zones and variables) is 1-based, so the last index is exactly equal to (not one less than) the count of the element type. An exception is nodes in finite-element zones, which can be specified to be zero-based. C and C++ programmers in particular should be wary of off-by-one errors when writing Tecplot data.

Zero is used in some functions to indicate the *lack* of a value; for example, if a zone is not partitioned, you pass zero as the Partition argument to `tecZoneVarWriteDoubleValues`.

Return Values and Errors

Return values in the API are used exclusively to indicate success or failure. Functions return zero on success and non-zero on failure.

Errors are often caused by file system or disk issues, such as insufficient space, or a programming error, such as calling a function before its prerequisite (e.g. trying to write data to a zone that hasn't yet been created).

Programming errors aside, if a TecIO function error occurs when writing a data file, writing cannot proceed further and should be aborted. You should attempt to close the file by calling `tecFileWriterClose`, but even if closing the file succeeds, the file may be in an incomplete state and may not be readable. You may offer to delete the file in this case, or leave this task for the user.

API Overview

The functions of the new TecIO writing API are divided into the following broad categories. Some functions are shown here in more than one category to make it easier to find the one you need.

- Initialization and File Handling
- Auxiliary Data
- Zones, Variables, and Data
- Partitioning and MPI
- Connectivity

Functions are generally listed alphabetically, except in cases where you will usually need a particular function first, or when two functions are closely related.

Initialization and File Handling

These functions open and close files, add top-level data elements such as auxiliary data and custom label sets, and initialize MPI. See Opening and Closing Files for details.

Function	Arguments
tecFileWriterOpen	Filename, Data Set Title, List of Variables[a], File Format[b], File Type, Default Data Type[c], Grid File Handle[d], File Handle[e]
tecFileWriterClose	File Handle
tecMPIInitialize	File Handle, MPI Communicator, Main Rank
tecDataSetAddAuxData	File Handle, Name, Value

a. A single string containing unquoted variable names separated by commas.
b. Currently, the new API supports only SZL (`.szplt`) files.
c. Use zero for the default data type. You must specify the data type for each variable when creating zones anyway.
d. From a previous call to tecFileWriterOpen that opened a grid file.
e. Output parameter that receives the new file's file handle.

Auxiliary Data

Auxiliary data records can be attached to three objects in a Tecplot file: the data set itself, a zone, and a variable. Each record is a name-value pair; both names and values are strings. The following functions are used to write auxiliary data items.

Function	Arguments
tecDataSetAddAuxData	File Handle, Name, Value
tecVarAddAuxData	File Handle, Variable Index, Name, Value
tecZoneAddAuxData	File Handle, Zone Index, Name, Value

Zones, Variables, and Data

Zones are used to organize data spatially, chronologically, or by some other characteristic. Variables are named collections of values with a data type and value location. These functions create zones and write variable values to zones. See Writing Data for details.

The TecIO API includes functions for writing face-based or "poly" (polygonal/polyhedral) zones. Currently, SZL files cannot contain such zones, so they are not implemented and are not documented here.

Function	Arguments
tecZoneCreateIJK[a]	File Handle, Zone Title, I Max, J Max, K Max, Variable Data Types[b], Variable Sharing Source Zones, Value Locations, Passive Variables, Face Neighbor Sharing Source Zone, Number of Face Connections, Face Neighbor Mode, Zone Index[c]
tecZoneCreateFE[d]	File Handle, Zone Title, Zone Type, Number of Nodes, Number of Cells, Variable Data Type[e], Variable Sharing Source Zones, Value Locations, Passive Variables, Connectivity Sharing Source Zone, Number of Face Connections, Face Neighbor Mode, Zone Index[f]
tecZoneCreateFEMixed	File Handle, Zone Title, Number of Nodes, Number of Sections, Cell Shape per Section, Grid Order per Section, Basis Function per Section, Number of Elements per Section, Variable Data Types, Variable Sharing Source Zones, Value Locations, Passive Variables, Connectivity Sharing Source Zone, Number of Face Neighbor Connections, Face Neighbor Mode, Zone Index
tecZoneSetUnsteadyOptions	File Handle, Zone Index, Solution Time, Strand ID
tecFileWriterFlush	File Handle[g], Number of Zones to Retain, Zones to Retain
tecZoneMapPartitionsToMPIRanks[h]	File Handle, Zone Index, Number of Partitions, Ranks for Partitions
tecZoneVarWriteDoubleValues[i] tecZoneVarWriteFloatValues tecZoneVarWriteInt32Values tecZoneVarWriteInt16Values TecZoneVarWriteUint8Values	File Handle, Zone Index, Variable Index, Partition Index, Number of Values, Array of Values

a. Variable Data Types, Variable Sharing Source Zones, Value Locations, and Passive Variables are integer arrays holding an element for

each variable, containing the specified argument value for the corresponding variable index.
b. Pass null for this parameter to use the default data type for all variables in the zone.
c. Output parameter that receives the index of the new zone.
d. See footnote a.
e. See footnote b.
f. See footnote c.
g. tecFileWriterFlush is identical to old API call (`TECFLUSH142`).
h. TecIO-MPI only.
i. These functions work the same; the only difference is the data type of the value array.

Partitioning and MPI

TecIO can subdivide zones, allowing each partition of a zone to be written independently (for example, by another thread, or by another process in the MPI-enabled version of TecIO). See Working With Partitions.

Function	Arguments
tecMPIInitialize[a]	File Handle, MPI Communicator, Main Rank
tecZoneMapPartitionsToMPIRanks[a]	File Handle, Zone Index, Number of Partitions, Ranks for Partitions
tecFEPartitionCreate32[b] tecFEPartitionCreate64	File Handle, Zone Index, Partition Index, Number of Nodes, Number of Cells, Number of Ghost Nodes, Array of Ghost Node Indexes, Array of Neighbor Partition Indexes, Array of Neighbor Partition Node Indexes, Number of Ghost Cells, Array of Ghost Cell Indexes
tecFEMixedPartitionCreate32[b] tecFEMixedPartitionCreate64	File Handle, Zone Index, Partition Index, Number of Nodes, Number of Cells per Section, Number of Ghost Nodes, Array of Ghost Node Indexes, Array of Neighbor Partition Indexes, Array of Neighbor Partition Node Indexes, Number of Ghost Cells per Section, Array of Ghost Cell Indexes
tecIJKPartitionCreate	File Handle, Zone Index, Partition Index, I Min, J Min, K Min, I Max, J Max, K Max

a. TecIO-MPI only.
b. These functions work the same way; the only difference is the size of the indexes.

Connectivity

Connectivity refers to how the nodes and cells in finite element zones are connected and encompasses the node map and face neighbors.

The TecIO API includes functions for writing connectivity information for face-based or "poly" (polygonal/polyhedral) zones. Currently, SZL files cannot contain such zones,

so these functions are not implemented and are not documented here.

Function	Input
tecZoneNodeMapWrite32[a] tecZoneNodeMapWrite64	File Handle, Zone Index, Partition Index, One-Based Node Flag (Boolean), Node Count, Array of Nodes
tecZoneFaceNbrWriteConnections32[b] tecZoneFaceNbrWriteConnections64	File Handle, Zone Index, Array of Face Neighbors

a. These functions work the same way; the only difference is the size of the node indexes.
b. These functions work the same way; the only difference is the size of the face neighbor indexes.

Opening and Closing Files

The functions `tecFileWriterOpen` and `tecFileWriterClose` are used to open and close files. `tecFileWriterOpen` passes back a file handle—a pointer to a data structure internal to the library that you should treat as opaque. This handle is a required parameter to virtually all other functions in this API.

You specify the data set's title and the names of the variables it will contain when you open a file. You also specify the default data type for variables stored in the file.

It is possible to open more than one file at a time, then pass the handle for the specific file you want to write to each API function. The number of simultaneously open files is limited only by memory and/or the underlying operating system.

A file may be a grid file, a solution file, or a single file containing both a grid and a solution. You specify which kind of file you are writing using the File Type argument in the `tecFileWriterOpen` call. If writing a solution file, you must specify the file handle of corresponding grid file, which must be data-complete (but still open) before you write any solution files. Any number of solution files may use the same grid file simultaneously; do not close the grid file until you have written all the solution files that use it. If writing a grid file, or a single file containing both solution and grid, pass null for the grid file handle.

You must close files when you are done writing them to ensure all data is flushed to disk. Unlike the close function in many APIs, `tecFileWriterClose` requires that you pass in a pointer to the variable where you originally stored the file handle when you called `tecFileWriterOpen`, *not the file handle itself*. TecIO will store a null pointer in this variable to help make sure you do not try to use the file handle again after closing the file. (This is necessary because file handles are guaranteed unique only as long as a file is open. Two files opened at different times might end up with the same handle.)

Working With Partitions

The TecIO library is available in two variants: TecIO, and TecIO-MPI. The two variants have nearly identical APIs. However, TecIO-MPI uses the Message-Passing Interface (MPI) library to allow a file to

be written by multiple processes.

Zones may be *partitioned*. That is, the data does not need to be written by a single process, but can be written in sections and the work divided among multiple processes. This capability is intended primarily for use with CFD codes that compute large solutions in parallel, with multiple solver processes running on independent nodes in a compute cluster, each solving a subset of the simulation. However, this capability can also be used by a single-process solver running on a desktop workstation if desired.

TecIO-MPI is required only for multi-processing applications. Partitioned zones may still be written with the non-MPI TecIO variant, but they must be written by a single application process. TecIO is not thread-safe, so writing must always be done by a single thread—either the main thread, or a separate writing thread.

To use MPI with a given file, you must call `tecMPIInitialize` after opening the file but before writing to it . This function tells TecIO which MPI communicator to use when writing the file, and which process ("rank," in MPI terms) is the main one. The main process is the only one that can write auxiliary data, text, geometries, and custom labels.

Overlap and Ghosts

Partitioned solution of large CFD cases typically requires that zone partitions overlap slightly. That is, the nodes or cells on the each side of the boundary between partitions will reside in more than one solver process: in the one that "owns" it according to the partitioning rules, as well as in any processes solving a partition spatially adjacent to it. In processes other than the one that "owns" them, such data are commonly referred to as "ghost cells" and "ghost nodes."

The TecIO library needs information about these overlapping cells and nodes in order to later allow the partitioned data to be joined into a single virtual data set suitable for visualization. For finite-element zones, each solver process must pass a list of its ghost nodes and cells along with the zone data (which should include the ghost nodes and cells). For ordered zones, the solver need pass only the nodes owned by each process, as the indices of each partition are sufficient for later reassembling the data.

Partitioning Zones

After creating a zone, you may partition it as follows.

- First, if you are using TecIO-MPI, call `tecZoneMapPartitionsToMPIRanks` to specify how many partitions will be in the zone, and which MPI rank each partition belongs to. (This is not necessary with non-MPI TecIO.)
- Then call `tecFEPartitionCreate32`, `tecFEPartitionCreate64`, or `tecIJKPartitionCreate` to create each partition in the zone. For FE zones, these function calls specify the ghost cells and nodes.

When writing data, you specify the partition index of each zone you are writing. If you have not called

any of the functions to create partitions, always pass 0 for this argument.

When writing an unpartitioned zone with TecIO-MPI, specify a single process in the `tecZoneMapPartitionsToMPIRanks` call. The zone will then be a non-partitioned zone written entirely by that process.

Writing Data

In a Tecplot binary data file, actual data elements are contained by zones and variables.

- Zones are used to organize data spatially, chronologically, or by some other characteristic. Each zone has a type (in a SZL file, finite element or IJK-ordered zones are supported) and a name. You create zones using `tecZoneCreateIJK` or `tecZoneCreateFE`.
- Variables are named collections of values with a data type and value location. You specify the names of variables when you first create the file using the Variable List argument to `tecFileWriterOpen`. You specify the data type, location (nodal or cell-centered) and other characteristics of each variable when creating individual zones, since these properties can vary from one zone to the next.

After creating a zone, which specifies settings for the variables in that zone, call one of the `tecZoneVarWrite` functions to pass the data for the zone. There is one such function for each value type (e.g. `tecZoneVarWriteFloatValues` for 32-bit floating point values). You don't necessarily need to call the one that matches the type of the variable in the file; where possible, TecIO converts the output data to the proper type (for example, you can pass 32-bit floating point values when writing a variable that consists of 64-bit floating-point values).

It is not necessary to write all the data at once. You use the Count parameter in the `tecZoneVarWrite` function to specify how many elements you are writing in that call. This way, you can easily write values as you have them, and don't need to keep them around any longer than you actually need them. See Writing IJK-Ordered Data and Writing Finite Element Data for specific notes about writing data to these types of zones.

Passive and Shared Variables

Not every zone might have values for every variable. A variable is specified to be *passive* if it does not exist at all in a given zone. (The value of such a variable is taken to be zero everywhere.)

A variable may also be *shared* if it is to have the same value in more than one zone. In finite-element solutions, the spatial variables X, Y, and Z are commonly shared among all zones.

Passive and shared variables are specified when creating the zone they belong to by passing:

- An array of Booleans that specifies which variables are passive.
- An array of integers specifiying the index of the zone from which the variable is shared, or 0 if the variable is not shared. The variable must not be a shared variable in the zone from which it is

shared.

Writing IJK-Ordered Data

The I, J, and K indexes describe the extent of an ordered zone in three dimensions. If the extent of one or more indexes is 1, the zone is two-dimensional or one-dimensional.

Even though the actual zone may have 1, 2, or 3 dimensions, the values array are passed one-dimensionally, with the I-index varying the fastest and the K index the slowest. That is, all the I indexes should be used before varying J, and all the J indexes should be used before varying K.

Writing Finite Element Data

A finite element (FE) zone is made up of a series of shapes, such as triangles or bricks, all having the same number of vertices. Variable values can be designated to exist at either the nodes (vertices) or in the center of the cells of each zone. Variable values in an FE zone are indexed by node number or cell number, not by spatial coordinates.

The location of each cell is determined by the values of the variables X, Y, and Z (or e.g. X and Y for a 2D FE zone), which are always nodal variables and so specify the spatial placement of the vertices of the zone's cells. The actual values for a variable in a FE zone are passed to TecIO as a list of the values of the variable for each node or each cell (depending on whether the zone designates the variable as nodal or cell-centered).

A separate data structure, the *node map,* specifies which vertices make up each cell, and must be provided for each FE zone. The node map consists of a set number of values for each cell of the zone. For example, a triangle zone contains three entries in the node map for each cell, because a triangle has three vertices. The first three entries in the node map specify the vertices that make up the first cell; the next three specify the vertices that make up the second cell, and so on. See Zone Type for information on the number of nodes required for each FE element shape.

As with the data values themselves, the node map need not be given to TecIO all at once. Using the Count argument in the `tecZoneNodeMapWrite32` or `64` function, you can specify the number of node map entries you are passing on each call. You can pass as few or as many as you want at a time, so long as the total number of node map entries you pass corresponds to the number of data elements you passed by the time you close the file.

The difference between these two functions is the size of the index, which is specified for the variable when you create the zone. If your variable contain more than about 2 billion values, you must use 64-bit indexing. If it will be smaller, using 32-bit indexing requires only half the space for the node map.

Writing Face Neighbors

Face neighbors are a list of cells indicating the cells that share faces. When loading data, Tecplot 360 or

Tecplot Focus automatically detects which cells share faces based on the IJK indexing of the cells (for indexed zones) or the entries in the node map (for finite element zones). If your data contains additional face neighbors that cannot be detected in this way, specify them using `tecZoneFaceNbrWriteConnections32` or `tecZoneFaceNbrWriteConnections64`.

The number of entries per face neighbor (in the face neighbors array passed to TecIO) is determined by the Face Neighbor Mode, passed when creating the zone. See `TECFACE142` in the classic API documentation for a discussion of how many entries are required for each mode.

Writing Auxiliary Data

Tecplot data files can store named values referred to as *auxiliary data*. Auxiliary data is often used to store metadata about a dataset or to hold the results of aggregate calculations such as overall lift or maximum and minimum pressure. In Tecplot data files, auxiliary data can be attached to three points: the dataset itself, the zone, and the variable. Each dataset, zone, and variable may have any number of auxiliary data records attached.

Writing these types of auxiliary data is straightforward with the functions `tecDataSetAddAuxData`, `tecVarAddAuxData`, and `tecZoneAddAuxData`. These functions are called similarly; for variable and zone auxiliary data, you must pass the variable or zone index, respectively.

Auxiliary data names should be unique for the object they are attached to—that is, you should not have two auxiliary data records named LIFT on a single zone; having LIFT on two different zones, or on a zone and the dataset (etc.), is fine.

Auxiliary data values must be strings. If you wish to represent numeric quantities or, for example, a comma-separated list of values or other record-style data, you must convert this to a string before writing.

Other Elements

It is possible to write the following data elements using the new writing API:

- Text objects
- Geometries (shapes) of various sorts
- Custom labels

Most data files do not need to include these types of elements (especially text, as the font selection is very limited for text elements in a data file). In general, we recommend creating these elements with a layout file *(.lay)* and providing that with your data file, especially for text objects (layouts can use any font installed on the system, while text objects in a data file can use only a limited solution of built-in fonts).

Example

The `rewriteszl` example is a simple program that reads a SZL data file using the TecIO SZL reading API (see Reading SZL Data Files) and writes it to a new SZL file using the new writing API. The resulting file is functionally equivalent to the original, though it may not be identical at the binary level.

As such, the code illustrates writing every kind of data that can be stored in a SZL file. It also prints information about each data object as it is read, which may make it helpful as a diagnostic tool. This example is provided in both C++ `(.cpp)` and FORTRAN `(.F, .f90)` versions. Visual Studio solution files are also provided.

SZCOMBINE

`szcombine` (`szcombine.exe` on Windows) is a command-line utility to combine into a .szplt file the six temporary files produced by calls to `TECFLUSH142` (or `tecFileWriterFlush` in the new API). It is installed in Tecplot 360's bin folder, and is also built when the Tecio library is built from the source package available from the tecplot.com website. Its syntax is as follows:

```
szcombine infileBaseName [--cleanup] [--outfile outFileName]
```

infileBaseName is the file name passed into the FName parameter of `TECINI142`, to which was added six suffixes to form the temporary file names. With no optional parameters, the temporary files are combined to produce a .szplt file whose name is infileBaseName, and temporary files are left in place. The --cleanup option causes the temporary files to be deleted. The --outfile option causes the .szplt file to be written to a different filename. For example, given the following temporary files that were produced by passing `myfile.szplt` into TECINI142:

```
myfile.szplt.szhdr  myfile.szplt.szdat  myfile.szplt.szaux
myfile.szplt.sztxt  myfile.szplt.szgeo  myfile.szplt.szlab
```

Then executing the command

```
szcombine myfile.szplt
```

would create myfile.szplt and leave the six temporary files in place. Note that if the program that produced the temporary files (AKA solver) is still running and calls `TECEND142`, this file will be overwritten by a new myfile.szplt that contains all data written by the solver.

Executing instead the command

```
szcombine myfile.szplt --cleanup
```

would create `myfile.szplt` and delete the temporary files. This is the same effect that calling `TECEND142` from the solver would have had. NOTE: Do not use the `--cleanup` option if the solver is still running. It will delete the temporary files that the solver expects to combine when it calls TECEND142.

Finally, executing the command

```
szcombine myfile.szplt --outfile mynewfile.szplt
```

Would create file `mynewfile.szplt` leave the six temporary files in place. You may also use the two optional arguments together to create `.szplt` file with a non-default name and delete the temporary files. Again, do not do this if the source program (solver) is still running.

Binary Data File Format

Refer to this section only if you wish to write your own code that writes Tecplot binary-format (`.plt`) files without utilizing the TecIO library. Otherwise, refer to Getting Started for instructions for linking with this library, which is provided at no cost by Tecplot, Inc.

Remember that polygonal zones are not available for Tecplot Focus. Refer to Creating Data Files for Tecplot 360 and Focus for details.

The subzone file format `(.szplt)` is not currently documented because we anticipate the need for changes to the file format in the near future. Furthermore, the format is sufficiently complex that we do not expect application developers finding it worthwhile to undertake writing their own code to output in `.szplt` format. If you want your application to write subzone files and find the TecIO library insufficient, please contact us.

```
/*
BINARY FILE FORMAT:
------------------------------------------------------------------------
Two binary data file formats are detailed below: v112 and v191. The differences
are minor. Preplot produces v112 which was introduced with Tecplot 11.2 in 2009.
v191 was introduced with Tecplot 2019r1. v191 supports more than 2 billion
faces and face-nodes for poly data, but is the same for all other types of data.

Binary data files have two main sections.  A header section and a data
section.


         +----------------+
         | HEADER SECTION |
         +----------------+
         +---------+
         | FLOAT32 |              EOHMARKER, value=357.0
```

```
        +---------+
        +----------------+
        | DATA SECTION   |
        +----------------+
```

```
I.  HEADER SECTION

    The header section contains: the version number of the file, a title
    of the file, the names of the variables to be plotted, the
    descriptions of all zones to be read in and all text and geometry
    definitions.

     i.  Magic number, Version number
         +-----------+
         | "#!TDVnnn"|       8 Bytes, exact characters "#!TDV112" or
         +-----------+       "#TDV191". The version number follows the
                             capital "V" and uses the next 3 characters
                             (for example: "V75 ", "V112" or "V191").
                             v112 files support only v112 zone headers.
                             v191 files support both v112 and v191
                             zone headers. (See below for zone headers.)

     ii. Integer value of 1.
         +-----------+
         | INT32     |       This is used to determine the byte order
         +-----------+       of the reader, relative to the writer.



     iii. Title and variable names.
         +-----------+
         | INT32     |       FileType: 0 = FULL,
         +-----------+                 1 = GRID,
                                       2 = SOLUTION
         +-----------+
         | INT32*N   |       The TITLE. (See note 1.)
         +-----------+
         +-----------+
         | INT32     |       Number of variables (NumVars) in the datafile.
         +-----------+
         +-----------+
         | INT32*N   |       Variable names.
         +-----------+          N =  L[1] + L[2] + .... L[NumVars]
                             where:
                                    L[i] = length of the ith variable name + 1
```

```
                                        (for the terminating 0 value).
                        (See note 1.)
    iv.  Zones
        +-----------+
        | FLOAT32   |       Zone marker. Value = 299.0 or 298.0
        +-----------+          This zone marker determines the zone header version.
                               299.0 indicates a v112 zone header (available in both
                               v112 and v191 file versions)
                               298.0 indicates a v191 zone header (available only
                               in v191 file version)
        +-----------+
        | INT32*N   |       Zone name. (See note 1.)
        +-----------+            N = (length of zone name) + 1.
        +-----------+
        | INT32     |       ParentZone: No longer used as of Tecplot version 2020r1
        +-----------+
        +-----------+
        | INT32     |       StrandID: -2 = pending strand ID for assignment
        +-----------+                     by Tecplot
                                    -1 = static strand ID
                                     >=0 valid strand ID
        +-----------+
        | FLOAT64   |       Solution time.
        +-----------+
        +-----------+
        | INT32     |       Default Zone Color. Seldom used.  Set to -1.
        +-----------+
        +-----------+
        | INT32     |       ZoneType 0=ORDERED,        1=FELINESEG,
        +-----------+                2=FETRIANGLE,     3=FEQUADRILATERAL,
                                     4=FETETRAHEDRON,  5=FEBRICK,
                                     6=FEPOLYGON,      7=FEPOLYHEDRON
        +-----------+
        | INT32     |       Specify Var Location.
        +-----------+          0 = Don't specify, all data is located
                                   at the nodes.
                               1 = Specify
        if "specify var location" == 1
          +-----------+
          | INT32*NV  |     Variable Location (only specify if above is 1).
          +-----------+     0 = Node, 1 = Cell Centered (See note 5.)
        +-----------+
        | INT32     |       Are raw local 1-to-1 face neighbors supplied?
        +-----------+       (0=FALSE 1=TRUE).   These raw values are a
                            compact form of the local 1-to-1 face neighbors.
                            If supplied, Tecplot assumes that the face
                            neighbors are fully specified.  As such, it
                            will not perform auto face neighbor assignment.
```

```
                      This improves Tecplot's time to first plot.
                      See the data section below for format details.
                      ORDERED and FELINESEG zones must specify 0 for
                      this value because raw face neighbors are not
                      defined for these zone types.  FEPOLYGON and
                      FEPOLYHEDRON zones must specify 0 for this value
                      since face neighbors are defined in the face map
                      for these zone types.

        +-----------+
        | INT32     |   Number of miscellaneous user-defined face
        +-----------+   neighbor connections (value >= 0). This value
                        is in addition to the face neighbors supplied
                        in the raw section.  FEPOLYGON and FEPOLYHEDRON
                        zones must specify 0.

        if "number of miscellaneous user-defined
            face neighbor connections" != 0
          +-----------+
          | INT32     |   User defined face neighbor mode
          +-----------+   (0=Local 1-to-1, 1=Local 1-to-many,
                          2=Global 1-to-1, 3=Global 1-to-many)
          if FE Zone:
            +-----------+
            | INT32     |   Indicates if the finite element face neighbors
            +-----------+   are completely specified by the miscellaneous
                            face neighbors given: (0=NO, 1=YES). If yes,
                            then Tecplot will not perform auto assignment
                            of face neighbors otherwise all faces not
                            specified are considered boundaries. If no,
                            then Tecplot will perform auto-assignment of
                            the face neighbors unless the raw face neighbor
                            array was supplied. This option is not valid
                            for ORDERED zones.
        if Ordered Zone:
          +-----------+
          | INT32*3   |   IMax,JMax,KMax
          +-----------+

        if FE Zone:
          +-----------+
          | INT32     |   NumPts
          +-----------+
          if ZoneType is FEPOLYGON or FEPOLYHEDRON:
            +-----------+
            | INT32/64  |   NumFaces: For v112 zone headers, this is an INT32.
            +-----------+     For v191 zone headers, this is an INT64.
            +-----------+
```

```
          | INT32/64  |   Total number of face nodes. For v112 zone headers
          +-----------+   this is an INT32. For v191 zone headers, this is an INT64.
                          (For FEPOLYGON zones, this is NumFaces*2.)
          +-----------+
          | INT32     |   Total number of boundary faces.  If any
          +-----------+   boundary faces exist, include one to represent
                          no neighboring element.
          +-----------+
          | INT32     |   Total number of boundary connections.
          +-----------+


        +-----------+
        | INT32     |     NumElements
        +-----------+
        +-----------+
        | INT32*3   |     ICellDim,JCellDim,
        +-----------+     KCellDim (for future use; set to zero)

     For all zone types (repeat for each Auxiliary data name/value pair):
     +-----------+
     | INT32     |       1=Auxiliary name/value pair to follow
     +-----------+       0=No more Auxiliary name/value pairs.

     If the above is 1, then supply the following:
       +-----------+
       | INT32*N   |     name string (See note 1.)
       +-----------+
       +-----------+
       | INT32     |     Auxiliary Value Format
       +-----------+     (Currently only allow 0=AuxDataType_String)

       +-----------+
       | INT32*N   |     Value string  (See note 1.)
       +-----------+


  v.  Geometries
     +-----------+
     | FLOAT32   |       Geometry marker.  Value = 399.0
     +-----------+
     +-----------+
     | INT32     |       Position CoordSys 0=Grid, 1=Frame,
     +-----------+                         2=FrameOffset(not used),
                                           3= OldWindow(not used),
                                           4=Grid3D
     +-----------+
     | INT32     |       Scope 0=Global 1=Local
     +-----------+
```

```
+-----------+
| INT32     |       DrawOrder 0=After, 1=Before
+-----------+
+-----------+
| FLOAT64*3 |       (X or Theta),(Y or R),(Z or dummy)
+-----------+            i.e. the starting location
+-----------+
| INT32     |       Zone (0=all)
+-----------+
+-----------+
| INT32     |       Color
+-----------+
+-----------+
| INT32     |       FillColor
+-----------+
+-----------+
| INT32     |       IsFilled (0=no 1=yes)
+-----------+
+-----------+
| INT32     |       GeomType  0=Line, 1=Rectangle 2=Square,
+-----------+                 3=Circle, 4=ellipse
+-----------+
| INT32     |       LinePattern  0=Solid 1=Dashed 2=DashDot
+-----------+                    3=DashDotDot 4=Dotted
                                 5=LongDash
+-----------+
| FLOAT64   |       Pattern Length
+-----------+
+-----------+
| FLOAT64   |       Line Thickness
+-----------+
+-----------+
| INT32     |       NumEllipsePts
+-----------+
+-----------+
| INT32     |       Arrowhead Style 0=Plain, 1=Filled, 2=Hollow
+-----------+
+-----------+
| INT32     |       Arrowhead Attachment 0=None, 1=Beg, 2=End, 3=Both
+-----------+
+-----------+
| FLOAT64   |       Arrowhead Size
+-----------+
+-----------+
| FLOAT64   |       Arrowhead Angle
+-----------+
+-----------+
| IN32*N    |       Macro Function Command (string: N = Length+1)
```

```
         +-----------+
         +-----------+
         | INT32     |       Polyline Field Data Type
         +-----------+          1=Float, 2=Double  (GTYPE)
         +-----------+
         | INT32     |       Clipping (0=ClipToAxes, 1=ClipToViewport,
         +-----------+                 2=ClipToFrame)


If the geometry type is line then:
         +-----------+
         | INT32     |       Number of polylines
         +-----------+
         +-----------+
         | INT32     |       Number of points, line 1.
         +-----------+
         +-----------+
         | GTYPE*N   |       X-block geometry points N=NumPts
         +-----------+
         +-----------+
         | GTYPE*N   |       Y-block geometry points N=NumPts
         +-----------+
         +-----------+
         | GTYPE*N   |       Z-block geometry points N=NumPts (Grid3D Only)
         +-----------+
             .
             .
             .

If the geometry type is Rectangle then
         +-----------+
         | GTYPE*2   |       X and Y offset for far corner of rectangle
         +-----------+

If the geometry type is Circle then
         +-----------+
         | GTYPE     |       Radius
         +-----------+

If the geometry type is Square then
         +-----------+
         | GTYPE     |       Width
         +-----------+

If the geometry type is Ellipse then
         +-----------+
         | GTYPE*2   |       X and Y Radii
         +-----------+
```

```
vi.   Text
     +-----------+
     | FLOAT32   |       Text marker.  Value=499.0
     +-----------+
     +-----------+
     | INT32     |       Position CoordSys 0=Grid, 1=Frame,
     +-----------+                       2=FrameOffset(not used),
                                         3= OldWindow(not used),
                                         4=Grid3D(New to V10)
     +-----------+
     | INT32     |       Scope 0=Global 1=Local
     +-----------+
     +-----------+
     | FLOAT64*3 |       (X or Theta),(Y or R),(Z or dummy)
     +-----------+               Starting Location
     +-----------+
     | INT32     |       FontType
     +-----------+
     +-----------+
     | INT32     |       Character Height Units 0=Grid, 1=Frame, 2=Point
     +-----------+
     +-----------+
     | FLOAT64   |       Height of characters
     +-----------+
     +-----------+
     | INT32     |       Text Box type 0=NoBox 1=Hollow 2=Filled
     +-----------+
     +-----------+
     | FLOAT64   |       Text Box Margin
     +-----------+
     +-----------+
     | FLOAT64   |       Text Box Margin Linewidth
     +-----------+
     +-----------+
     | INT32     |       Text Box Outline Color
     +-----------+
     +-----------+
     | INT32     |       Text Box Fill Color
     +-----------+
     +-----------+
     | FLOAT64   |       Angle
     +-----------+
     +-----------+
     | FLOAT64   |       Line Spacing
     +-----------+
     +-----------+
```

```
        | INT32     |       Text Anchor. 0=left,      1=center,
        +-----------+                    2=right,     3=midleft
                                         4=midcenter  5=midright,
                                         6=headleft   7=headcenter
                                         8=headright
        +-----------+
        | INT32     |       Zone (0=all)
        +-----------+
        +-----------+
        | INT32     |       Color
        +-----------+
        +-----------+
        | INT32*N   |       MacroFunctionCommand (string: N = Length + 1)
        +-----------+
        +-----------+
        | INT32     |       Clipping (0=ClipToAxes,
        +-----------+                 1=ClipToViewport, 2=ClipToFrame)
        +-----------+
        | INT32*N   |       Text.  N=Text Length+1
        +-----------+

     vii.CustomLabel
        +-----------+
        | FLOAT32   |       CustomLabel Marker;  F=599
        +-----------+
        +-----------+
        | INT32     |       Number of labels
        +-----------+
        +-----------+
        | INT32*N   |       Text for label 1.  (N=length of label + 1)
        +-----------+           See note 1.
        +-----------+
        | INT32*N   |       Text for label 2.  (N=length of label + 1)
        +-----------+           See note 1.
            .
            .
            .
        +-----------+
        | INT32*N   |       Text for label NumLabels.
        +-----------+           (N=length of label + 1) See note 1.

    viii.UserRec
        +-----------+
        | FLOAT32   |       UserRec Marker;  F=699
        +-----------+
        +-----------+
        | INT32*N   |       Text for UserRec.  See note 1.
        +-----------+
```

```
    ix. Dataset Auxiliary data.
         +-----------+
         | FLOAT32   |       DataSetAux Marker;  F=799.0
         +-----------+
         +-----------+
         | INT32*N   |       Text for Auxiliary "Name".  See note 1.
         +-----------+
         +-----------+
         | INT32     |       Auxiliary Value Format (Currently only
         +-----------+       allow 0=AuxDataType_String)
         +-----------+
         | INT32*N   |       Text for Auxiliary "Value".  See note 1.
         +-----------+
    x. Variable Auxiliary data.
         +-----------+
         | FLOAT32   |       VarAux Marker;  F=899.0
         +-----------+
         +-----------+
         | INT32*N   |       Variable number (zero based value)
         +-----------+
         +-----------+
         | INT32*N   |       Text for Auxiliary "Name".  See note 1.
         +-----------+
         +-----------+
         | INT32     |       Auxiliary Value Format (Currently only
         +-----------+       allow 0=AuxDataType_String)
         +-----------+
         | INT32*N   |       Text for Auxiliary "Value".  See note 1.
         +-----------+

II.  DATA SECTION (don't forget to separate the header from the data
     with an EOHMARKER).  The data section contains all of the data
     associated with the zone definitions in the header.

     i. For both ordered and fe zones:
         +-----------+
         | FLOAT32   |       Zone marker  Value = 299.0 if this zone was
         +-----------+         defined by a v112 zone header. 298.0 if this
                               zone was defined by a v191 zone header.
         +-----------+
         | INT32*N   |       Variable data format, N=Total number of vars
         +-----------+           1=Float,    2=Double, 3=LongInt,
                                 4=ShortInt, 5=Byte,   6=Bit
         +-----------+
         | INT32     |       Has passive variables: 0 = no, 1 = yes.
         +-----------+
         if "has passive variables" != 0
```

```
      +-----------+
      | INT32*NV  |     Is variable passive: 0 = no, 1 = yes
      +-----------+     (Omit entirely if "Has passive variables" is 0).
    +-----------+
    | INT32     |       Has variable sharing 0 = no, 1 = yes.
    +-----------+
    if "has variable sharing" != 0
      +-----------+
      | INT32*NV  |     Zero based zone number to share variable with
      +-----------+     (relative to this datafile). (-1 = no sharing).
                        (Omit entirely if "Has variable sharing" is 0).
    +-----------+
    | INT32     |       Zero based zone number to share connectivity
    +-----------+       list with (-1 = no sharing). FEPOLYGON and
                        FEPOLYHEDRON zones use this zone number to
                        share face map data.


    Compressed list of min/max pairs for each non-shared and non-passive
    variable. For each non-shared and non-passive variable (as specified
    above):
      +-----------+
      | FLOAT64   |       Min value
      +-----------+
      +-----------+
      | FLOAT64   |       Max value
      +-----------+
    +-----------+
    | xxxxxxxxxx|       Zone Data.  Each variable is in data format as
    +-----------+       specified above.

  ii. specific to ordered zones
    if "zone number to share connectivity list with" == -1 &&
       "num of misc. user defined face neighbor connections" != 0
      +-----------+
      | INT32*N   |     Face neighbor connections.
      +-----------+     N = (number of miscellaneous user defined
                             face neighbor connections) * P
                        (See note 5 below).


  iii. specific to fe zones when FileType is not SOLUTION:
    if ZoneType is NOT FEPOLYGON or FEPOLYHEDRON :
      if "zone number to share connectivity lists with" == -1
        +-----------+
        | INT32*N   |     Zone Connectivity Data N=L*JMax
        +-----------+     (see note 2 below ).
```

```
      if "zone number to share connectivity lists with" == -1 &&
         "raw local 1-to-1 face neighbors are supplied"
        +-----------+
        | INT32*N   |     Raw local 1-to-1 face neighbor array.
        +-----------+     N = (NumElements * NumFacesPerElement)
                          (See note 3 below).

      if "zone number to share connectivity lists with" == -1 &&
         "num of misc. user defined face neighbor connections" != 0
        +-----------+
        | INT32*N   |     Face neighbor connections.
        +-----------+     N = (number of miscellaneous user defined
                               face neighbor connections) * P
                          (See note 4 below).

    if ZoneType is FEPOLYGON or FEPOLYHEDRON:
      if "zone number to share face map data with" == -1
        if the zone header version for this zone is v112
          +-----------+
          |INT32*(F+1)|     Face node offsets into the face nodes array
          +-----------+     below. Does not exist for FEPOLYGON zones.
                            F = NumFaces.
        otherwise the zone header version for this zone is v191
          +-----------+
          | INT32*F   |     Face node count for each face.
          +-----------+     Does not exist for FEPOLYGON zones.
                            F = NumFaces.

        +-----------+
        | INT32*FN  |     Face nodes array containing the node numbers
        +-----------+     for all nodes in all faces.
                          FN = total number of face nodes.

        +-----------+
        | INT32*F   |     Elements on the left side of all faces.
        +-----------+     Boundary faces use a negative value which is
                          the negated offset into the face boundary
                          connection offsets array.  A value of "-1"
                          indicates there is no left element.
                          F = NumFaces.

        +-----------+
        | INT32*F   |     Elements on the right side of all faces.  See
        +-----------+     description of left elements above for more
                          details.  F = NumFaces.

        if "total number of boundary faces" != 0
          +-----------+
```

```
               | INT32*NBF |    Boundary face connection offsets into the
               +-----------+    boundary face connecion elements array and
                                the boundary face connection zones array.
                                The number of elements for a face (F) is
                                determined by offset[-o] - offset[-o-1]
                                where 'o' is the negative value from either
                                the left or right elements arrays above.
                                Offset[0] = 0. Offset[1] = 0 so that -1 as
                                the left or right element always indicates
                                no neighboring element.  If the number of
                                elements is 0, then there is no neighboring
                                element.
                                NBF = total number of boundary faces + 1.

               +-----------+
               | INT32*NBI |    Boundary face connection elements.  A value of
               +-----------+    "-1" indicates there is no element on part of
                                the face.
                                NBI = total number of boundary connections.

               +-----------+
               | INT32*NBI |    Boundary face connection zones.  A value of
               +-----------+    "-1" indicates the current zone.
                                NBI = total number of boundary connections.


NOTES:

1.  All character data is represented by INT32 values.

     Example:  The letter "A" has an ASCII value of 65.  The WORD
               written to the data file for the letter "A" is then
               65. In fortran this could be done by doing the following:

               Integer*32 I
               .
               .
               I = ICHAR('A');

               WRITE(10) I


    All character strings are null terminated
       (i.e. terminated by a zero value)


2.  This represents JMax sets of adjacency zero based indices where each
    set contains L values and L is
```

```
    2 for LINESEGS
    3 for TRIANGLES
    4 for QUADRILATERALS
    4 for TETRAHEDRONS
    8 for BRICKS


3.  The raw face neighbor array is dimensioned by (number of elements for
    the zone) times (the number of faces per element), where each member
    of the array holds the zero-based element neighbor of that face. A
    boundary face is one that has no neighboring element and is
    represented by a -1. Faces should only be neighbors if they logically
    share nodes and they should be reciprocal.


4.  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 (zero based)
        fz = face of cell in current zone (zero based)
        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 (zero based)
        CZ = cell in remote zone (zero based)

    cz,fz combinations must be unique and multiple entries are
    not allowed. 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.

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

5.  Cell centered variable (DATA SECTION)
    To make reading of cell centered binary data efficient, Tecplot stores
    IMax*JMax*(KMax-1) numbers of cell centered values, where IMax, JMax,
    and KMax represent the number of points in the I, J, and K directions.
    Therefore extra zero values (ghost values) are written to the data file
    for the fastest moving indices. For example, if your data's IJK
```

```
    dimensions are 2x3x2, a cell-centered variable will have 1x2x1
    (i.e. (I-1)x(J-1)x(K-1)) significant values. However, 2x3x1 values must
    be written out because it must include the ghost values. Assume that the
    two significant cell-centered values are 1.5 and 12.5. The ghost values
    will be output with a zero value.

    So if the zone was dimensioned 2x3x2 its cell centered variable would be
    represented as follows:
      1.5   0.0  12.5   0.0   0.0   0.0

    If the zone was dimensioned 3x2x2 its cell centered variable would be
    represented as follows:
      1.5  12.5   0.0   0.0   0.0   0.0

    and if the zone was dimensioned 2x2x3 its cell centered variable would be
    represented as follows:
      1.5   0.0   0.0   0.0  12.5   0.0   0.0   0.0

    For large variables the wasted space is less significant than it
    is for the small example above.
-------------------------------------------------------------------------
*/
```

Meaningful Integer Values

The TecIO library often uses integers with special meanings, for example for colors, fonts, and zone types. The same values are used both for writing (TECXXXX142 functions) and for SZL reading and writing (tecXxxXxx functions). The table below documents the meaning of these values. Where available, the equivalent keywords used in Tecplot ASCII files are also included.

Category	Legal Values	ASCII Keyword	API Functions
Boolean	0: False 1: True		Many
File Format	0: Tecplot binary (.plt) 1: Tecplot subzone (.szplt)	n/a	TECINI1142 tecFileWriterOpen
File Type	0: Full (grid and solution) 1: Grid only 2: Solution only	FULL GRID SOLUTION	TECINI1142 tecFileGetType tecFileWriterOpen
Arrowhead Attachment	0: None 1: Beginning 2: End 3: Both beginning and end	NONE BEGINNING END BOTH	TECGEO142 tecGeomArrowheadGetAttach tecGeomArrowheadSetInfo

Category	Legal Values	ASCII Keyword	API Functions
Arrowhead Style	0: Plain 1: Filled 2: Hollow	PLAIN FILLED HOLLOW	TECGEO142 tecGeomArrowheadGetStyle tecGeomArrowheadSetInfo
Clipping	0: Clip to viewport 1: Clip to frame	CLIPTOVIEWPORT CLIPTOFRAME	TECGEO142, TECTXT142 tecGeomGetClipping tecGeomSetClipping tecTextGetClipping tecTextSetClipping
Color	0: Black 1: Red 2 :Green 3: Blue 4: Cyan 5: Yellow 6: Purple 7: White 8–15 - Custom 1–8	BLACK RED GREEN BLUE CYAN YELLOW PURPLE WHITE CUST1–CUST8	TECGEO142, TECTXT142 tecGeomFill tecGeomGetColor tecGeomGetFillColor tecGeomSetLineInfo tecTextBoxGetColor tecTextBoxGetFillColor tecTextBoxSetInfo tecTextGetColor tecTextSetColor
Coordinate Mode	0: Grid 1: Frame 6: 3D Grid[a]	GRID FRAME GRID3D	TECGEO142 tecGeom2DLineSegmentsBegin tecGeom2DMultiLineSegmentsBegin tecGeomCircleBegin tecGeomEllipseBegin tecGeomGetCoordMode tecGeomRectangleBegin tecGeomSquareBegin tecText2DBegin tecTextGetCoordMode
Data Type[b]	1: Single-precision (4) 2: Double-precision (8) 3: 32-bit integer (4) 4: 16-bit integer (2) 5: 8-bit unsigned int (1)	SINGLE DOUBLE	TECINI142 tecZoneVarGetType tecZoneCreateIJK tecZoneCreateFE tecZoneCreateFEMixed tecFileWriterOpen
Face Neighbor Mode	0: Local one-to-one 1: Local one-to-many 2: Global one-to-one 3: Global one-to-many	LOCALONETOONE LOCALONETOMANY GLOBALONETOONE GLOBALONETOMANY	TECZNE142 TECZNEFEMIXED142 tecZoneFaceNbrGetMode tecZoneCreateIJK tecZoneCreateFE tecZoneCreateFEMixed

Category	Legal Values	ASCII Keyword	API Functions
Geometry Type	0: 2D polyline 1: Rectangle 2: Square 3: Circle 4: Ellipse 5: 3D polyline	LINE SQUARE RECTANGLE CIRCLE ELLIPSE LINE3D	TECGEO142 tecGeomGetType
Line Pattern	0: Solid 1: Dashed 2: Dash Dot 3: Dotted 4: Long Dash 5: Dash Dot Dot	SOLID DASHED DASHDOT DOTTED LONGDASH DASHDOTDOT	TECGEO142 tecGeomGetLinePattern tecGeomSetLineInfo
Scope	0: Local—active frame only 1: Global—all frames that share this data set	LOCAL GLOBAL	TECGEO142, TECTXT142 tecGeomGetScope tecGeomSetScope tecTextGetScope tecTextSetScope
Text Anchor	0: Bottom left 1: Bottom center 2: Bottom right 3: Middle left 4: Middle center 5: Middle right 6: Top left 7: Top center 8: Top right	LEFT CENTER RIGHT MIDLEFT MIDCENTER MIDRIGHT HEADLEFT HEADCENTER HEADRIGHT	TECTXT142 tecTextGetAnchor tecTextSetAnchor
Text Box Type	0: None 1: Filled (opaque) 2: Hollow (outline only)	NOBOX FILLED HOLLOW	TECTXT142 tecTextBoxGetType tecTextBoxSetInfo
Text Size Units	0: Grid units 1: % of frame height 2: Typographical points	GRID FRAME POINTS	TECTXT142 tecText2DBegin tecText3DBegin tecTextGetSizeUnits

Category	Legal Values	ASCII Keyword	API Functions

Typeface/Font[c]	0: Helvetica
	1: Helvetica Bold
	2: Greek
	3: Math
	4: User-Defined
	5: Times
	6: Times Italic
	7: Times Bold
	8: Times Italic Bold
	9: Courier
	10: Courier Bold
		HELV
		HELV-BOLD
		GREEK
		MATH
		USER-DEF
		TIMES
		TIMES-ITALIC
		TIMES-BOLD
		TIMES-ITALIC-BOLD
		COURIER
		COURIER-BOLD
			TECTTXT142
			`tecTextGetTypeface`
			`tecTextSetTypeface`

Value Location	0: Cell-centered
	1: Nodal
		CELLCENTERED
		NODAL
			TECZNE142
			TECZNEFEMIXED142
			TECPOLYZNE142
			`tecZoneVarGetValueLocation`
			`tecZoneCreateIJK`
			`tecZoneCreateFE`
			`tecZoneCreateFEMixed`

Zone Type[d]	0: Ordered
	1: FE line segment (2)
	2: FE triangle (3)
	3: FE quadrilateral (4)
	4: FE tetrahedron (4)
	5: FE brick (8)
	6: FE polygon (variable)
	7: FE polyhedron (variable)
	8: FE Mixed (variable)
		ORDERED
		FELINESEG
		FETRIANGLE
		FEQUADRILATERAL
		FETETRAHEDRON
		FEBRICK
		FEPOLYGON
		FEPOLYHEDRAL
		(not used)
			TECZNE142
			TECPOLYZNE142[e]
			`tecZoneCreateFE`
			`tecZoneGetType`

a. Only for 3D line segment geometries and text objects. Text with this coordinate mode is visible only in 3D Cartesian plots.
b. When writing data using the classic API, you may choose only between single-precision and double-precision float types via a Boolean flag in TECINI142. However, the new writing API (along with Tecplot products and add-ons) can create zones of other types. When reading files, you should be prepared to find different types—some possibly integers. Number of parentheses is the number of bytes per element.
c. Due to the limited font selection available in data files, you will usually want to specify text objects in layouts or stylesheets, where any font installed on the system may be specified by name.
d. Number in parentheses is the number of nodes per element.
e. FEPOLYGON and FEPOLYHEDRAL only.

See the following chapters for more information on how these values are used:

- Binary Data

- ASCII Data
- Reading SZL Data Files
- Writing SZL Data Files (New API)

Glossary

The following terms are used throughout the Data Format Guide and are included here for your reference.

2D	Plotting in two dimensions. Line plots of one or more variables (XY and Polar Line plots) are not considered 2D.
2D Cartesian Plot	A plot of some variable by location on a single plane using two axes.
3D	Plotting in three dimensions. Three-dimensional plotting can be subdivided into 3D surface and 3D volume.
3D Cartesian Plot	A plot displaying a 3D scattering of points, surfaces, or volumes using three orthogonal axes.
3D Surface	Three-dimensional plotting confined to a surface. For example, the surface of a wing.
3D Volume	Three-dimensional plotting of data that includes interior data points of a volume, as well as those on the surface. For example, the vector field around a wing.
Active Zone	A zone that is displayed in the current plot, as determined in the Zone Style dialog.
ASCII Data File	A data file composed of human-readable statements and numbers using ASCII characters.
Auxiliary Data	Metadata attached to zones, data sets, and frames.
Binary Data File	A data file composed of machine-readable data. This type of file is created by converting ASCII data files with Preplot, or by directly creating them from an application.
Block	A data file format in which the data is listed by variable. All the point values of the first variable are listed first, then all the point values of the second variable, and so forth.
Boundary Cell Faces	A set of un-blanked cell faces in a 3D volume zone which have only one neighboring volume cell. In contrast, interior cell faces have two neighboring volume cells, one on either side, which share the face. For an IJK-ordered zone the boundary cell faces are on the exterior of the zone. That is, the first and last I-planes, the first and last J-planes, and the first and last K-planes. For a finite element 3D volume zone, boundary cell faces are on the exterior of the zone and the surface of any voids within the zone.

Brick	An element type of finite element volume data composed of eight node points arranged in a hexahedron-like format. This element type is used in 3D volume plotting.
Cell	Either an element of finite element data, or the space contained by one increment of each index of IJ- or IJK-ordered data.
Cell-Centered Values	Values located at the center of the cell (assumed to be the centroid).
Connectivity List	The portion of a finite element data file which defines the elements or cells by listing the relationships between points. The number of points per cell is determined by the element type.
Custom Labels	Text strings contained within a data file or text geometry file which define labels for your axes or contour table. You may select Custom Labels anywhere you can choose a number format, the result is the text strings in place of numbers. The maximum length of a custom label is 1024 characters.
Data File	A file that contains data used for plotting in Tecplot.
Data Format	The type of zone data as specified by the format parameter in a Tecplot data file, such as: BLOCK or POINT.
Data Loader	A Tecplot add-on which allows you to read non-Tecplot data files.
Data Point	An XYZ-point at which field variables are defined.
Data Set	A set of one or more zones. A data set may be plotted in one or more frames. However, a single frame may only plot one data set. A data set may be created by loading one or more data files.
Element Type	The form of individual elements in a finite element zone. There are four types of cell-based finite element zones: Triangle and Quadrilateral (finite element surface types), and Tetrahedron and Brick (finite element volume types). For cell-based finite elements, the element type of a zone determines the number of nodes per element and their orientation within an element. There are two types of face-based finite element zones: polygonal (2D) and polyhedral (3D). For face-based elements, the number of nodes per element is variable. Tecplot Focus cannot load face-based zones.
FE	An abbreviation for finite element, a common means of arranging data for calculations. (Often referred to as "unordered" or "unstructured".)
FE Surface	A finite element zone of the element type Triangle, Quadrilateral, Polygon. These zones are used for 2D and 3D surface plots.
FE Volume	A finite element zone of the element type Tetrahedron, Brick, Polyhedron. These zones are used for 3D volume plots.
Field Map	A collection of zones for 2D and 3D field plots. A common style can be easily applied to all zones in the selection.

Field Plot	Includes 2D Cartesian and 3D Cartesian plot types. Generally used to display the spacial relationship of data. Mesh, Contour, Vector, Scatter and Shade are all considered field plots. XY and Polar Line plots and the Sketch plot type are not field plots.
Finite Element	A type of data point ordering. Data is arranged by listing the data points (called nodes), and then listing their relationships (called elements). The element type of the zone determines the number of nodes which are contained in each element, as well as the exact relationship of nodes within an element. There are several different element types supported by Tecplot: Triangle, Quadrilateral, Tetrahedron, Brick, Polygonal, and Polyhedral, See also: Connectivity List, and Node,
I-Ordered	A type of data point ordering where each point is listed one at a time (that is, by one index). Used mainly in XY-plots. In 2D or 3D, this type of data point ordering is sometimes called irregular, and is only useful for scatter plots, or for interpolating into 2D, 3D surface, or 3D volume zones. (This type of data can also be used for 2D or 3D vector plots if streamtraces are not required.)
IJ-Ordered	A type of data point ordering where the points are arranged in a 2D array used for 2D and 3D surface plotting.
IJK-Blanking	A feature to include or exclude portions of an IJK-ordered zone based on index ranges.
IJK-Ordered	A type of data ordering where the points are arranged in a 3D array. Used for 3D volume plotting as well as 2D and 3D surface plotting.
I-Plane	In an ordered zone, the connected surface of all points with a constant I-index. In reality, I-planes may be cylinders, spheres, or any other shape.
Irregular Data	Points which have no order, or at least no order which can be easily converted to IJ- or IJK-ordering.
J-Plane	In an ordered zone, the connected surface of all points with a constant J-index. In reality, J-planes may be cylinders, spheres, or any other shape.
K-Plane	In an IJK-ordered zone, the connected surface of all points with a constant K-index. In reality, K-planes may be cylinders, spheres, or any other shape.
Macro	A file containing a list of instructions, called macro commands, which can duplicate virtually any action performed in Tecplot.
Macro Command	An instruction given to Tecplot in a macro file. Macro commands always start with a dollar sign and then an exclamation mark. For example, $!Redraw refreshes a plot view.
Macro File	A file which contains a series of macro commands. Macro files are run from the command line, or through the Play option of the Macro sub-menu of the File menu.
Macro Function	A self-contained macro sub-routine.

Macro Variable	A holding place for numeric values in a macro file. There are two types of macro variables: user-defined (you set and retrieve the value), or internal (Tecplot sets the value and you may retrieve it).
No Neighboring Element	In polyhedral/polygonal fe data sets, the term "no neighboring element" refers to a face that does not have a neighboring element on either its right or left side.
Node	A point in finite element data.
Number Format	The style of numbers to display for a data or axis label; exponent, integer, float, and so forth.
Ordered Data	A type of data point organization which consists of a parameterized series of points. There are seven types of ordered data: I-, J-, K-, IJ-, JK-, IK-, and IJK-ordered. I-, IJ-, and IJK-ordered are the most common.
Polygonal	A 2D, face-based finite element type. The number of nodes per element is variable. That is, a single polygonal zone may contain triangular, quadrilateral, hexagonal, ..., etc. elements. Tecplot Focus cannot load this type of zone.
Polyhedral	A 3D, face-based finite element type. The number of nodes per element is variable. That is, a single polyhedral zone may contain tetrahedral and brick (and others) elements. Tecplot Focus cannot load this type of zone.
Point	A data file format for an I-, IJ-, or IJK-ordered zone in which the data is listed by point. All of the variable values for the first data point are listed first, then all the variable values for the second data point, and so forth.
Quadrilateral	An element type of finite element surface data which is composed of four node points arranged in a quadrilateral. Used in 2D and 3D surface plotting.
Sharing	Variable sharing allows a single storage location to be used by more than one party. For example, if the X-variable is shared between zones five and seven only one storage location is created. The storage is not freed by Tecplot until the number of parties accessing the data is reduced to zero. Variables and connectivity information may be shared.
Subzone	A portion of a zone,. Also refers to a file format (`.szplt`) introduced in Tecplot 360 EX that allows subzones to be loaded as needed for plots and other operations. In typical use cases, this significantly improves interactive performance and reduces memory footprint compared to `.plt` files.
Tetrahedron	An element type of finite element volume data which is composed of four node points arranged in a tetrahedron. (Used in 3D volume plotting.)
Triangle	An element type of finite element surface data which is composed of three node points arranged in a triangle. (Used in 2D and 3D surface plotting.)
Unordered or Unorganized Data	(See Irregular Data)

Zone	A subset of a data set which is assigned certain plot types. Zones may be activated (plotted) or deactivated (not plotted). Each zone has one type of data ordering: I-, IJ-, IJK-, or finite element. Zones are typically used to distinguish different portions of the data. For example, different calculations, experimental versus theoretical results, different time steps, or different types of objects, such as a wing surface versus a vector field around a wing.
Zone Layers	One way of displaying a 2D or 3D plot's data set. The plot is the sum of the active zone layers, which may include mesh, contour, vector, shade, scatter and edge.

Copyright

Tecplot 360 Data Format Guide is for use with Tecplot 360 2025 R2.

Tecplot, Inc.
Post Office Box 52708
Bellevue, WA 98015-2708 U.S.A.
Tel:1.800.763.7005 (within the U.S. or Canada), 00 1 (425) 653-1200 (internationally)
E-mail: sales@tecplot.com, support@tecplot.com
Questions, comments or concerns regarding this document: support@tecplot.com
For more information, visit www.tecplot.com

Part Number: 23-360-05-2 Build Revision 85213

Released: 2/2026

For third-party trademark and copyright information, see the User's Manual.