



TECPLOT 360™
2008

Data Format Guide

COPYRIGHT NOTICE

Tecplot 360™ Data Format Guide is for use with Tecplot 360™ Version 2008.

Copyright © 1988-2008 Tecplot, Inc. All rights reserved worldwide. Except for personal use, this manual may not be reproduced, transmitted, transcribed, stored in a retrieval system, or translated in any form, in whole or in part, without the express written permission of Tecplot, Inc., 3535 Factoria Blvd, Ste. 550, Bellevue, WA 98006 U.S.A.

The software discussed in this documentation and the documentation itself are furnished under license for utilization and duplication *only* according to the license terms. The copyright for the software is held by Tecplot, Inc. Documentation is provided for information only. It is subject to change without notice. It should not be interpreted as a commitment by Tecplot, Inc. Tecplot, Inc. assumes no liability or responsibility for documentation errors or inaccuracies.

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)
email: sales@tecplot.com, support@tecplot.com
Questions, comments or concerns regarding this document: documentation@tecplot.com
For more information, visit <http://www.tecplot.com>

THIRD PARTY SOFTWARE COPYRIGHT NOTICES

SciPy © 2001-2002 Enthought, Inc. All Rights Reserved. NumPy © 2005 NumPy Developers. All Rights Reserved. VisTools and VdmTools © 1992-2007 Visual Kinematics, Inc. All Rights Reserved. NCSA HDF & HDF5 (Hierarchical Data Format) Software Library and Utilities © 1988-2004 The Board of Trustees of the University of Illinois. Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC), Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratories (SNL), Los Alamos National Laboratory (LANL), Jean-I All Rights Reserved. PNG Reference Library © Copyright (c) 1995, 1996 Guy Eric Schlatn, Group 42, Inc., Copyright (c) 1996, 1997 Andreas Dilger, Copyright (c) 1998, 1999 Glenn Randers-Pehrson. All Rights Reserved. Tcl © 1989-1994 The Regents of the University of California. Copyright © 1994 The Australian National University. Copyright © 1994-1998 Sun Microsystems, Inc. Copyright © 1998-1999 Scrips Corporation. All Rights Reserved. bmtopm © 1992 David W. Sanderson. All Rights Reserved. Nephm © 1988 Jef Poskanzer. All Rights Reserved. Mesa © 1999-2001 Brian Paul. All Rights Reserved. W3C IPR © 1995-1998 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University. All Rights Reserved. Ppmtopic © 1990 Ken Yap. All Rights Reserved. JPEG © 1991-1998 Thomas G. Lane. All Rights Reserved.

TRADEMARKS

Tecplot®, Tecplot 360™, the Tecplot 360™ logo, Preplot™, Enjoy the View™, and Framer™ are registered trademarks or trademarks of Tecplot, Inc. in the United States and other countries.

3D Systems is a registered trademark or trademark of 3D Systems Corporation in the U.S. and/or other countries. Macintosh OS is a registered trademark or trademark of Apple, Incorporated in the U.S. and/or other countries. Reflection-X is a registered trademark or trademark of Attachmate Corporation in the U.S. and/or other countries. LAPACK is a registered trademark or trademark of The University of Tennessee in the U.S. and/or other countries. EnSight is a registered trademark or trademark of Computation Engineering International (CEI), Incorporated in the U.S. and/or other countries. DEM is a registered trademark or trademark of DEM Solution Ltd in the U.S. and/or other countries. Exceed 3D and Hummingbird Exceed is a registered trademark or trademark of Hummingbird Limited in the U.S. and/or other countries. Konqueror is a registered trademark or trademark of KDE e.V in the U.S. and/or other countries. VIP and VDB are registered trademarks or trademarks of Landmark Graphics Corporation in the U.S. and/or other countries. ECLIPSE FrontSim are registered trademarks or trademarks of Schlumberger, Limited in the U.S. and/or other countries. Debian is a registered trademark or trademark of Software in the Public Interest in the U.S. and/or other countries. X3D is a registered trademark or trademark of Web3D Consortium in the U.S. and/or other countries. X Windows is a registered trademark or trademark of X Consortium, Incorporated in the U.S. and/or other countries. ANSYS, Fluent and any and all ANSYS, Inc. brand, product, service and feature names, logos and slogans is a registered trademark or trademark of ANSYS Incorporated or its subsidiaries in the U.S. and/or other countries. ESIPAM - CRASH is a registered trademark or trademark of ESI Group in the U.S. and/or other countries. LSTC/DYNA is a registered trademark or trademark of Livermore Software Technology Corporation in the U.S. and/or other countries. MSC/NASTRAN is a registered trademark or trademark of MSC Software Corporation in the U.S. and/or other countries. NASTRAN is a registered trademark or trademark of National Aeronautics Space Administration in the U.S. and/or other countries. JDSL is a registered trademark or trademark of STREAMSim Technologies in the U.S. and/or other countries. SDR/IDEAS Universal is a registered trademark or trademark of UGS PLM Solutions Incorporated or its subsidiaries in the U.S. and/or other countries. SciPy is a registered trademark or trademark of Enthought, Inc in the U.S. and/or other countries. NumPy is a registered trademark or trademark of NumPy Developers in the U.S. and/or other countries. Star-CCM+ is a registered trademark or trademark of CD-adapco in the U.S. and/or other countries. VDB are registered trademarks or trademarks of Landmark Graphics Corporation in the U.S. and/or other countries. FLEXnet is a registered trademark or trademark of Macrovision Corporation and/or Macrovision Europe Ltd in the U.S. and/or other countries. Python is a registered trademark or trademark of Python Software Foundation in the U.S. and/or other countries. Visual Kinematics, Inc in the U.S. and/or other countries. Ultimate Grid 97 is a registered trademark or trademark of DUNDAS Software, Ltd in the U.S. and/or other countries. Xbae is a registered trademark or trademark of Copyright (c) 1991, 1992 Bell Communications Research, Inc. (Bellcore), Copyright (c) 1995-99 Andrew Lister. Copyright © 1999 - 2004 by the LessTie/Xbae maintenance team in the U.S. and/or other countries. NCSA HDF & HDF5 (Hierarchical Data Format) Software Library and Utilities is a registered trademark or trademark of the Board of Trustees of the University of Illinois in the U.S. and/or other countries. Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC), Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratories (SNL), Los Alamos National Laboratory (LANL), Jean-Ibaqus, the 3DS logo, SIMULIA and CATIA is a registered trademark or trademark of Dassault Systemes in the U.S. and/or other countries. The Abaqus runtime libraries are a product of Dassault Systemes Simulia Corp., Providence, RI, USA. FLOW3D is a registered trademark or trademark of Flow Science, Incorporated in the U.S. and/or other countries. Adobe, Flash, Flash Player, Premier and PostScript is a registered trademark or trademark of Adobe Systems, Incorporated in the U.S. and/or other countries. AutoCAD and DXF is a registered trademark or trademark of Autodesk, Incorporated in the U.S. and/or other countries. Ubuntu is a registered trademark or trademark of Canonical, Limited in the U.S. and/or other countries. HP, LaserJet and PaintJet is a registered trademark or trademark of Hewlett-Packard Company in the U.S. and/or other countries. IBM, RS/6000 and AIX is a registered trademark or trademark of International Business Machines Corporation in the U.S. and/or other countries. Helvetica Font Family and Times Font Family is a registered trademark or trademark of Linotype GmbH in the U.S. and/or other countries. Linux is a registered trademark or trademark of Linus Torvalds in the U.S. and/or other countries. ActiveX, Excel, Microsoft, Visual C++, Visual Studio, Windows, Windows Metafile, Windows XP, Windows Vista, Windows 2000 and PowerPoint is a registered trademark or trademark of Microsoft Corporation in the U.S. and/or other countries. Firefox is a registered trademark or trademark of Mozilla Corporation in the U.S. and/or other countries. Netscape is a registered trademark or trademark of Netscape Communications Corporation in the U.S. and/or other countries. SUSE is a registered trademark or trademark of Novell, Incorporated in the U.S. and/or other countries. Red Hat is a registered trademark or trademark of Red Hat, Incorporated in the U.S. and/or other countries. SPARC is a registered trademark or trademark of SPARC International, Incorporated in the U.S. and/or other countries. Courier Font Family is a registered trademark or trademark of The Monotype Corporation in the U.S. and/or other countries. UNIX and Motif is a registered trademark or trademark of The Open Group in the U.S. and/or other countries. Qt is a registered trademark or trademark of Trolltech in the U.S. and/or other countries. Zlib is a registered trademark or trademark of Jean-loup Gailly and Mark Adler in the U.S. and/or other countries. OpenGL is a registered trademark or trademark of Silicon Graphics, Incorporated in the U.S. and/or other countries. xpm is a registered trademark or trademark of GROUPE BULL. in the U.S. and/or other countries. fplib is a registered trademark or trademark of Thomas Pfau in the U.S. and/or other countries. PNG Reference Library is a registered trademark or trademark of Copyright (c) 1995, 1996 Guy Eric Schlatn, Group 42, Inc., Copyright (c) 1996, 1997 Andreas Dilger, Copyright (c) 1998, 1999 Glenn Randers-Pehrson in the U.S. and/or other countries. Tcl is a registered trademark or trademark of The Regents of the University of California. Copyright © 1994 The Australian National University. Copyright © 1994-1998 Sun Microsystems, Inc. Copyright © 1998-1999 Scrips Corporation in the U.S. and/or other countries. glt is a registered trademark or trademark of Stephane Rehel in the U.S. and/or other countries. bmtopm is a registered trademark or trademark of David W. Sanderson in the U.S. and/or other countries. Nephm is a registered trademark or trademark of Jef Poskanzer in the U.S. and/or other countries. pthreads is a registered trademark or trademark of John E. Bossum. Copyright(C) 1999,2002 Pthreads-win32 contributors in the U.S. and/or other countries. Copyright(C) 1999,2004 Pthreads-win32 contributors Mesa is a registered trademark or trademark of Brian Paul in the U.S. and/or other countries. W3C IPR is a registered trademark or trademark of World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University in the U.S. and/or other countries. Ppmtopic is a registered trademark or trademark of Ken Yap in the U.S. and/or other countries. freetype is a registered trademark or trademark of David Turner, Robert Wilhelm, and Werner Lemberg in the U.S. and/or other countries. Motif is a registered trademark or trademark of Open Software Foundation, Inc in the U.S. and/or other countries. © Copyright 1996-2000 The Open Group and others/JPEG is a registered trademark or trademark of Thomas G. Lane in the U.S. and/or other countries.

All other product names mentioned herein are trademarks or registered trademarks of their respective owners.

NOTICE TO U.S. GOVERNMENT END-USERS

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and/or in similar or successor clauses in the DOD or NASA FAR Supplement. Contractor/manufacturer is Tecplot, Inc., 3535 Factoria Blvd, Ste. 550, Bellevue, WA 98006 U.S.A.

08-360-05-1

Rev 01/2008

Table of Contents

<i>Chapter 1</i>	<i>Introduction.....</i>	5
	Creating Data Files for Both Tecplot 360 & Tecplot Focus6 Best Practices	6
<i>Chapter 2</i>	<i>Binary Data.....</i>	9
	Getting Started.....	9
	Viewing Your Output	10
	Binary Function Notes	11
	<i>Deprecated Binary Functions</i>	11
	<i>Character Strings in FORTRAN.....</i>	11
	<i>Boolean Flags.....</i>	12
	Binary Data File Function Calling Sequence.....	12
	Writing to Multiple Binary Data Files	12
	Linking with the TecIO Library	13
	<i>UNIX/Linux/Macintosh:</i>	13
	<i>Windows:.....</i>	13
	Binary Data File Function Reference.....	14
	Defining Polyhedral and Polygonal Data.....	48
	<i>Boundary Faces and Boundary Connections</i>	48
	<i>FaceNodeCounts and FaceNodes.....</i>	50
	<i>FaceRightElems and FaceLeftElems.....</i>	52
	<i>FaceBoundaryConnectionElements and Zones.....</i>	54
	Examples	54
	<i>Face Neighbors.....</i>	54
	<i>Polygonal Example.....</i>	65
	<i>Multiple Polyhedral Zones</i>	73



Multiple Polygonal Zones..... 92
Polyhedral Example..... 113
IJ-ordered zone 118
Switching between two files..... 121
Text Example..... 126
Geometry Example..... 128

Chapter 3 *ASCII Data*..... 133

Preplot 133
Syntax Rules & Limits..... 133
ASCII File Structure 134
 File Header..... 135
 Zone Record..... 137
 Text Record 150
 Geometry Record..... 153
 Custom Labels Record..... 157
 Data Set Auxiliary Data Record..... 158
 Variable Auxiliary Data Record..... 159
 ASCII Data File Parameter Assignment Values 159
Ordered Data 160
 I-Ordered Data 160
 IJ-Ordered Data..... 160
 IJK-Ordered Data..... 161
 Ordered Data Examples..... 161
Finite-Element Data 170
 Variable and Connectivity List Sharing..... 172
 Finite-Element Data Set Examples 174
ASCII Data File Conversion to Binary 189
 Preplot Options..... 189
 Preplot Examples..... 189

Chapter 4 *Glossary* 191

Appendix A *Binary Data File Format* 197



Chapter 1 *Introduction*

Tecplot 360 can read in data produced in many different formats, one of which is its own native format. Refer to [Chapter 22 “Creating a Data Loader”](#) in the [ADK User’s Manual](#) for information on creating a data loader add-on for use with Tecplot 360.

This manual describes how to output your data into the Tecplot 360 data format. The following chapters are included in the manual:

- [Chapter 2 “Binary Data”](#) - Refer to this chapter for details on outputting data into Tecplot 360’s binary file format (*.plt). The chapter also includes instructions for linking with the TecIO library (a library of functions used to create binary data, included in your distribution). Refer to the final section in the chapter for detailed examples.
- [Chapter 3 “ASCII Data”](#) - We strongly recommend that you create binary data files. However, the ASCII data chapter is provided to allow you to create simple data files.
- [Chapter 4 “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 [Section 1 - 2 “Best Practices”](#).



1 - 1 Creating Data Files for Both Tecplot 360 & Tecplot 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, you need to be aware that polyhedral/polygonal zones are not supported in 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 cell-based finite-element zones (triangular, quadrilateral, tetrahedral or brick elements).

1 - 2 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.

1. Create Binary Data Files instead of ASCII

All else being equal, 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 included with your Tecplot distribution. To create ASCII files, you can write-out plain text using standard write statements.

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

- Your data files are small.
- Your application runs on a platform for which the **TecIO** library is not provided. Even if this is the case, please contact us at support@tecplot.com. There may be a way to resolve this issue.



Tecplot 360 includes a utility called Preplot which allows you to convert an ASCII file into a binary file. See [Section B - 4 “Preplot”](#) in the [User’s Manual](#) for more information on how to use Preplot.

2. Use Block Format instead of Point Format



Block format is by far the most efficient format when it comes to loading the file into Tecplot 360. If your data files are small and you can only obtain the data in a point-like format (e.g. with a spreadsheet), then using point format is acceptable.



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

3. Use the Native Byte Ordering for the Target Machine

When you create binary data, you can elect to produce these files in either Motorola byte order or Intel byte order. Tecplot 360 automatically detects the byte order and loads both types. However, it is more efficient if you produce files using the byte order used on the platform where you run Tecplot 360. For example if you produce a binary file on an SGI platform and then transfer the data to a Windows® platform or Intel-based Linux box, you should set the flag to reverse the bytes when generating the binary data file. See the notes about this option in [Section B - 4 “Preplot”](#) in the [User’s Manual](#) for the Preplot flag.

4. 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 [Section “TECZAUXSTR111”](#) on page 41 or [Section 3- 3.6 “Data Set Auxiliary Data Record”](#) for information on working with auxiliary data in binary or ASCII data files, respectively. For a list of auxiliary data names, see [Chapter 9 “Using Standardized Auxiliary Data”](#) in the [ADK User’s Manual](#).

5. Data Sharing

Share variables whenever possible. Variable sharing is commonly used for the spatial variables (X, Y, and Z) when you have many sets of data that use the same basic grid. This saves 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 [Section “TECZNE111”](#) on page 42 (for binary data) or [Section 3- 5.1 “Variable and Connectivity List Sharing”](#) (for ASCII data).

6. Passive Variables



Tecplot 360 can manage many datasets at the same time. However, within a given dataset you must supply the same number of variables for each zone. In some cases you may have data where there are many variables and, for some of the zones some of those variables are not important. If that is the case, you can 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.



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, and to debug programs.

Data files for Tecplot 360 are commonly created as output from an application program. These files are most often in ASCII format, and are then converted to a binary format with Preplot (see [Section 3 - 1 “Preplot”](#) for additional information).

To output your data into Tecplot’s binary format, you may use the static library provided with your Tecplot 360 installation or you may write your own binary functions. If you wish to write your own functions, refer to [Appendix A “Binary Data File Format”](#) for details on the structure of Tecplot’s binary file format. If you wish to link with the library provided by Tecplot, begin with [Section 2 - 1 “Getting Started”](#) and use [Appendix A “Binary Data File Format”](#) for reference.

2 - 1 Getting Started

Your Tecplot 360 distribution includes a library of utility functions that you can link with your application to create binary data files directly, bypassing the use of ASCII files. This allows for fewer files to manage, conserves disk space, and saves the time required to convert the files.

On UNIX®, Linux®, Macintosh® platforms, the utility functions discussed in [Section 2 - 7 “Binary Data File Function Reference”](#) are available in the library archive **tecio.a** which is located in the **lib** directory below the *\$TEC_360_2008* Directory. On Windows platforms, this library is called **TecIO.lib** and is located in the **bin** sub-directory of your installation.

When preparing to output your data in Tecplot’s binary format using the **TecIO** library, we recommend you perform the following steps:

1. Review [Section 2 - 4](#) and [Section 2 - 5](#) of this manual.



2. Review the example files provided in the *util/tecio* directory of your Tecplot installation. The example programs demonstrate the use of the **TecIO** utility functions and are provided in both FORTRAN and C:
 - **simtest.f**, **simtest.f90**, **simtest.c** - Demonstrates simple use of the **TecIO** utility functions.
 - **comtest.f**, **comtest.f90**, **comtest.c** - Demonstrates the complex use of **TecIO** utility functions such as multiple file generation and transient data.
3. Follow the instructions in [Section 2 - 6 “Linking with the TecIO Library”](#) for information on linking with the **TecIO** library.
4. Begin developing your code.

2 - 2 Viewing Your Output

You may load your binary files in Tecplot using the Tecplot Data loader (refer to [Section 4 - 15 “Tecplot-Format Loader”](#) for details). In addition, you may view information about your data file using any of the following techniques:

- [Pltview](#) - Pltview is a command line utility that displays the header information for your file. It is installed in *\$TEC_360_2008/bin*. Refer to [Section B - 6 “Pltview” on page 693](#) in the [User’s Manual](#) for details on working with pltview.
- [View Binary](#) - The **ViewBinary** add-on allows you to view the information in a Tecplot binary data (*.plt*) file. It is included in a standard Tecplot distribution. Refer to [Section 33- 3.19 “View Binary” on page 672](#) in the [User’s Manual](#) for details.
- [Data Set Information](#) dialog - You may use the **Data Set Information** dialog (accessed via the **Data** menu) to display information about your plt 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 [Section 6 - 3 “Data Set Information” on page 174](#) in 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 [Section 20 - 12 “Data Spreadsheet”](#) in the [User’s Manual](#) for details.



2 - 3 Binary Function Notes



The *.plt file that you create will be compatible with the version of Tecplot tied to the version of the TecIO library that you use. For example, if you use the TecIO library that was bundled with Tecplot 360 Version 2006, your files can be loaded with Tecplot 360 Version 2006 and newer.

This is independent of the version number used for the binary functions (e.g. the 111 in TECZNE111). For example, even if you use 110 functions with the version of the TecIO library included with this distribution, your plt file will be compatible with this version of Tecplot and newer.

2- 3.1 Deprecated Binary Functions

Functions that end in 110 or less are deprecated. We recommend you use the 111 binary function family.

The following functions were altered during the upgrade to the 111 family:

- **TECINI** - The FileType parameter was added **TECINI**. Files from previous versions are of type "FULL". See [Section "TECINI111"](#) on page 29 for additional information.
- **TECZNE** - Three parameters, TotalNumFaceNodes, NumConnectedBoundaryFaces and TotalNumBoundaryConnections were added to TECZNE111. Refer to [Section "TECZNE111"](#) on page 42 for details.

If you update existing binary function calls to use version 111, you will need to update all of your binary calls.

2- 3.2 Character Strings in FORTRAN

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

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

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



2- 3.3 Boolean Flags

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

2 - 4 Binary Data File Function Calling Sequence

For a given file, the binary data file functions must be called in a specific order.

The order is as follows:

[TECFOREIGN111](#) (Optional)

[TECINI111](#)

For each call to [TECINI111](#), use one or more of the following commands:

[TECAUXSTR111](#) (Optional)

[TECVAUXSTR111](#) (Optional)

[TECZNE111](#) (One or more to create multiple zones)

For each call to [TECZNE111](#), use one of more of these commands:

[TECDAT111](#) (One or more to fill each zone)

[TECNOD111](#) (One for each finite-element zone)

[TECFACE111](#) (One for each zone with face connections)

[TECPOLY111](#) (Optional - use for polyhedral data)

[TECZAUXSTR111](#) (Optional)

[TECLAB111](#) (Optional)

[TECGEO111](#) (Optional)

[TECTXT111](#) (Optional)

[TECFIL111](#) (Optional - use if you are switching between files)

[TECUSR111](#) (Optional)

[TECEND111](#)

Section 2 - 5 “Writing to Multiple Binary Data Files” explains how you can use the [TECFIL111](#) function along with the above functions to write to multiple files simultaneously.

2 - 5 Writing to Multiple Binary Data Files

Each time **TECINI111** is called it sets up a new file “context.” For each file context you must maintain the order of the calls as described in the previous section. The **TECFIL111** function is used to switch between file contexts. Up to 10 files can be written to at a time. **TECFIL111** can be called almost anywhere after **TECINI111** has been called. The only parameter to **TECFIL111**, an



integer, n , shifts the file context to the n th open file. The files are numbered relative to the order of the calls to **TECINI111**.

2 - 6 Linking with the TecIO Library

To output data in Tecplot's binary format, you may write your own functions or use the library provided with your installation. On Windows platforms, `tecio.lib` is installed in the `bin` directory of your Tecplot 360 installation.¹ On UNIX, Linux, Macintosh platforms, `tecio.a` is installed in the `lib` directory of your Tecplot 360 installation. Follow the instructions below to link with Tecplot's library.



The *.plt file that you create will be compatible with the version of Tecplot tied to the version of the TecIO library that you use. For example, if you use the TecIO library that was bundled with Tecplot 360 Version 2006, your files can be loaded with Tecplot 360 Version 2006 and newer.

This is independent of the version number used for the binary functions (e.g. the 111 in **TECZNE111**). For example, even if you use 110 functions with the version of the TecIO library included with this distribution, your plt file will be compatible with this version of Tecplot and newer.

2- 6.1 UNIX/Linux/Macintosh:

NOTE: Some f90 compilers do not accept the f90 file extension. You may need to rename the files and edit the Make script to build these examples.

1. Verify that `tecio.a` is located in the `lib` directory below the Tecplot home directory.
2. Set your `$TEC_360_2008` environment variable to the Tecplot home directory.
3. Run **Make**. (Capital M)

2- 6.2 Windows:

NOTE: Only the `.c` and `.f90` source files are used on Windows operating systems.

To link with the **TecIO** library, perform the following steps:

¹ On Windows platforms, you will need to include `tecio.dll` in any distributions you create. `Tecio.dll` is provided in `$TEC_360_2008/bin` along with `tecio.lib`.



1. Create a development project
2. List `$(STEC_360_2008)/bin/tecio.lib` as an additional dependency. In Visual Studio® 2005, this is accomplished via: **Configuration Properties>Linker>Input** in the Project Properties dialog.
3. Include the **TecIO** header files (`TECIO.h` and `TECXXX.h`), located in:
`STEC_360_2008/Include`.

Notes for Windows Programmers using Fortran:

The included project files were developed and tested with Compaq Visual Fortran version 6.6. File `tecio.f90` contains both Fortran-90 interfaces for all **TecIO** functions and some compiler-specific directives (the `!MSS$ATTRIBUTES` lines) to direct Visual Fortran to use `STDCALL` calling conventions with by-reference parameter passing.

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.

2 - 7 Binary Data File Function Reference

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

TECAUXSTR111

Writes auxiliary data for the data set to the data file. The function may be called any time between [TECINI111](#) and [TECEND111](#). 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).

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECAUXSTR111 (Name ,  
&                               Value)  
CHARACTER* (*) Name  
CHARACTER* (*) Value
```

C Syntax:

```
#include TECIO.h
```



```
INTEGER4 TECAUXSTR111 ( 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. NOTE: It must be a null-terminated character string and cannot contain spaces.
Value	The value to assign to the named auxiliary data. NOTE: 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");

TECAUXSTR111("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.

TECDAT111

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 [TECZNE111](#)).

TECDAT111 allows you to write your data in a piecemeal fashion in case it is not contained in one contiguous block in your program. **TECDAT111** must be called enough times to ensure that the correct number of values are written for each zone and that the aggregate order for the data is correct.



In the above summary, NumVars is based on the number of variable names supplied in a previous call to [TECINI111](#).

FORTRAN Syntax:

```
INTEGER*4 FUNCTION          TECDAT111 ( N,  
&                           Data,  
&                           IsDouble)  
INTEGER*4                   N  
REAL or DOUBLE PRECISION   Data(1)  
INTEGER*4                   IsDouble
```

C Syntax:

```
#include TECIO.h  
INTEGER4 TECDAT111 (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 - 1 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 (IsBlock and VarLocation are parameters supplied to [TECZNE111](#)):

Table 2 - 1: Data Arrangement

Zone Type	Var. Location	IsBlock	Number of Values	Order
Ordered	Nodal	1	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	Nodal	0	IMax* JMax* KMax* NumVars	Vars varies fastest, then I, then J, then K. That is, the numbers should be supplied in the following order: for (K=1;K<=KMax;K++) for (J=1;J<=JMax;J++) for (I=1;I<=IMax;I++) for(Var=1;Var<=NumVars;Var++) Data[Var, I, J, K] = value;



Table 2 - 1: Data Arrangement

Zone Type	Var. Location	IsBlock	Number of Values	Order
Ordered	Cell Centered	1 ^a	(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	1	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	Nodal	0	IMax (i.e. NumNodes) * NumVars	Vars varies fastest, then N. That is, the numbers should be supplied in the following order: for (N=1;N<=NumNodes;N++) for (Var=1;Var<=NumVars;Var++) Data[Var, N] = value;
Finite-element	Cell Centered	1 ^a	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;

a. Cell-centered data must be supplied in block format (i.e. IsBlock = 1 for all cell-centered data).

Example

Refer to the following examples in [Section 2 - 9 “Examples”](#) for examples using **TECDAT111**:

- [Section 2- 9.1 “Face Neighbors”](#)
- [Section 2- 9.2 “Polygonal Example”](#)



- [Section 2- 9.3 “Multiple Polyhedral Zones”](#)
- [Section 2- 9.4 “Multiple Polygonal Zones”](#)
- [Section 2- 9.5 “Polyhedral Example”](#)
- [Section 2- 9.6 “IJ-ordered zone”](#)

TECEND111

Must be called to close out the current data file. There must be one call to **TECEND111** for each [TECINI111](#).

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECEND111 ()
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECEND111 ();
```

Return Value:

0 if successful, -1 if unsuccessful.

Parameters:

None.

TECFACE111

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 may 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.



This function must be called after [TECNOD111](#), and may only be called if a non-zero value of *NumFaceConnections* was used in the previous call to [TECZNE111](#).

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECFACE111 (FaceConnections)
INTEGER*4                      FACECONNECTIONS (*)
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECFACE111 (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 <i>FaceNeighborMode</i> parameter (set via TECZNE111) 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 defined in [Figure 2-1](#).

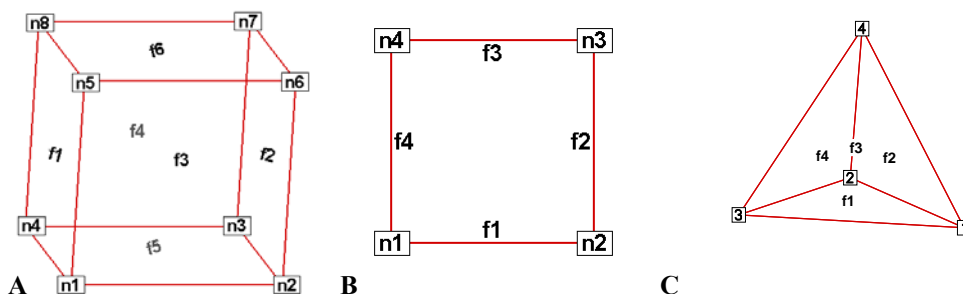


Figure 2-1. **A:** Example of node and face neighbors for an fe-brick cell or IJK-ordered cell. **B:** Example of node and face numbering for an IJ-ordered cell. **C:** example of tetrahedron face neighbors.

Example

Refer to [Section 2- 9.1 “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.

Switch output context to a different file. Each time [TECINI111](#) is called the file context is switched to a different file. This allows you to write multiple data files at the same time. When working with



multiple files, be sure to call **TECFIL111** each time you wish to write to a file. This will ensure your data is written to the appropriate file.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECFIL111 (F)
INTEGER*4                                F
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECFIL111 (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 TECINI111 .

Examples

Refer to [Section 2- 9.7 “Switching between two files”](#) for a simple example of working with **TECFIL111**.

TECFOREIGN111

Optional function that sets the byte ordering request for subsequent calls to [TECINI111](#). The byte ordering request will remain in effect until the next call to this function. This has no effect on files already opened via [TECINI111](#). Use this function to reverse the byte ordering from the format native to your operating system. For example, this is useful if you are creating a file on an SGI machine to be used on a Windows or Intel-based Linux machine. If the function call is omitted, native byte ordering will be used.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECFOREIGN111 (DoForeignByteOrder)
```



INTEGER*4

DoForeignByteOrder

C Syntax:

```
#include TECIO.h
INTEGER4 TECFOREIGN111(INTEGER4 *DoForeignByteOrder);
```

Return Value:

0 if successful, -1 if unsuccessful.

Parameters:

Parameter	Description
DoForeignByteOrder	Pointer to boolean value indicating if future files created by TECINI111 should be written out in foreign byte order. 0 indicates native byte order. 1 indicates foreign byte order.

TECGEO111

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

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECGEO111( XOrThetaPos,
&                               YOrRPos,
&                               ZPos,
&                               PosCoordMode,
&                               AttachToZone,
&                               Zone,
&                               Color,
&                               FillColor,
&                               IsFilled,
&                               GeomType,
&                               LinePattern,
&                               PatternLength,
&                               LineThickness,
&                               NumEllipsePts,
&                               ArrowheadStyle,
```



```

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

```

C Syntax:

```

#include TECIO.h
INTEGER4 TECGEO111(double *XOrThetaPos,

```




```
double *YOrRPos,  
double *ZPos,  
INTEGER4 *PosCoordMode,  
INTEGER4 *AttachToZone,  
INTEGER4 *Zone,  
INTEGER4 *Color,  
INTEGER4 *FillColor,  
INTEGER4 *IsFilled,  
INTEGER4 *GeomType,  
INTEGER4 *LinePattern,  
double *PatternLength,  
double *LineThickness,  
INTEGER4 *NumEllipsePts,  
INTEGER4 *ArrowheadStyle,  
INTEGER4 *ArrowheadAttachment,  
double *ArrowheadSize,  
double *ArrowheadAngle,  
INTEGER4 *Scope,  
INTEGER4 *Clipping,  
INTEGER4 *NumSegments,  
INTEGER4 *NumSegPts,  
float *XOrThetaGeomData,  
float *YOrRGeomData,  
float *ZGeomData,  
char *MFC
```

Return Value:

0 if successful, -1 if unsuccessful.



Parameter	Description
GeomType	<p>Pointer to integer value specifying the geometry type.</p> <p>0=2D Line Segments 3=Circle 1=Rectangle 4=Ellipse 2=Square 5=3D Line Segments</p>
LinePattern	<p>Pointer to integer value specifying the line pattern.</p> <p>0=Solid 3=Dotted 1=Dashed 4=LongDash 2=DashDot 5=DashDotDot</p>
PatternLength	<p>Pointer to double value specifying the pattern length in frame units (from 0 to 100).</p>
LineThickness	<p>Pointer to double value specifying the line thickness in frame units. The value must be greater than zero and less than 100.</p>
NumEllipsePts	<p>Pointer to integer value specifying the number of points to use for circles and ellipses. The value must be between 2 and 720.</p>
ArrowheadStyle	<p>Pointer to integer value specifying the arrowhead style.</p> <p>0=Plain 2=Hollow 1=Filled</p>
ArrowheadAttachment	<p>Pointer to integer value specifying where to attach arrowheads.</p> <p>0=None 2=End 1=Beginning 3=Both</p>
ArrowheadSize	<p>Pointer to double value specifying the arrowhead size in frame units (from 0 to 100).</p>
ArrowheadAngle	<p>Pointer to double value specifying the arrowhead angle in degrees.</p>
Scope	<p>Pointer to integer value specifying the scope with respect to frames. A local scope places the object in the current frame. A global scope places the object in all frames that contain the current frame's data set.</p> <p>0=Global 1=Local.</p>
Clipping	<p>Specifies whether to clip the geometry (that is, only plot the geometry within) to the viewport or the frame.</p> <p>0=ClipToViewport 1=ClipToFrame.</p>
NumSegments	<p>Pointer to integer value specifying the number of polyline segments.</p>
NumSegPts	<p>Array of integer values specifying the number of points in each of the NumSegments segments.</p>



Parameter	Description
XOrThetaGeomData	Array of floating-point values specifying the X-, Y- and Z-coordinates. Refer to “Data Values” on page 28 for information regarding the values required for each GeomType .
YOrRGeomData	
ZGeomData	
MFC	Macro function command. Must be null terminated.

Examples

Refer to [Section 2- 9.9 “Geometry Example”](#) for a simple example of working with **TECGEO111**.

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**.

TECINI111

Initializes the process of writing a binary data file. This must be called *first* before any other **TecIO** calls are made (except [TECFOREIGN111](#)). You may write to multiple files by calling **TECINI111** more than once. Each time **TECINI111** is called, a new file is opened. Use [TECFIL111](#) to switch between files. For each call to **TECINI**, there must be a corresponding call to [TECEND111](#).

FORTRAN Syntax:

```

INTEGER*4 FUNCTION TECINI111( Title,
&                               Variables,
&                               FName,
&                               ScratchDir,
&                               FileType,
&                               Debug,
&                               VIsDouble)
CHARACTER* ( *)                Title
CHARACTER* ( *)                Variables
CHARACTER* ( *)                ScratchDir
CHARACTER* ( *)                FName
INTEGER*4                      FileType
INTEGER*4                      Debug
INTEGER*4                      VIsDouble

```

C Syntax:

```

#include TECIO.h
INTEGER4 TECINI111(char      *Title,
                    char      *Variables,
                    char      *FName,
                    char      *ScratchDir,
                    INTEGER4  *FileType,

```



```

INTEGER4 *Debug
INTEGER4 *VIsDouble) ;

```

Return Value:

0 if successful, -1 if unsuccessful.

Parameters:

Parameter	Description
Title	Title of the data set. <i>Must be null terminated.</i>
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. <i>Must be null terminated.</i>
FName	Name of the file to create. <i>Must be null terminated.</i>
ScratchDir	Name of the directory to put the scratch file. <i>Must be null terminated.</i>
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 TECDAT111 are to be written in single or double precision. 0=Single 1=Double.

Examples

Each example in [Section 2 - 9 “Examples”](#) calls **TECINI111** at least once. Refer to this section for details.

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. NOTE: To work with custom labels, you must have at least one zone in your data set. A custom label set is added to your file each time you



call **TECLAB111**. 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.

Custom labels are assigned to an object via the Tecplot interface. Refer to [Section 18- 6.1 “Using Custom Labels”](#) in the [User’s Manual](#) for details.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECLAB111 (Labels)
CHARACTER* (*)           Labels
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECLAB111(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 TECLAB111 .

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\"";
TECLAB111 (&Labels[0]);
```

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 [TECPOLY111](#).



FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECNOD111 (NData)
INTEGER*4 NData (T, M)
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECNOD111 (INTEGER4 *NData);
```

Return Value:

0 if successful, -1 if unsuccessful.

Parameters:

Parameter	Description						
NData	<p>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:</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td>2=Line Segment</td> <td>4=Tetrahedral</td> </tr> <tr> <td>3=Triangle</td> <td>8=Brick</td> </tr> <tr> <td>4=Quadrilateral</td> <td></td> </tr> </table>	2=Line Segment	4=Tetrahedral	3=Triangle	8=Brick	4=Quadrilateral	
2=Line Segment	4=Tetrahedral						
3=Triangle	8=Brick						
4=Quadrilateral							

Examples:

Refer to [Section 2-9.1 “Face Neighbors”](#) for examples using **TECNOD111**.

TECPOLY111

Writes the face map for polygonal and polyhedral zones to the data file. All numbering schemes are one-based. That is, the first node is Node 1, the first face is Face 1, and so forth. Refer to [Section 2-8 “Defining Polyhedral and Polygonal Data”](#) on page 48 for additional information.

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

FORTRAN syntax:

```
INTEGER*4 FUNCTION TECPOLY111 (
&                               FaceNodeCounts,
```



```

&          FaceNodes ,
&          FaceLeftElems ,
&          FaceRightElems ,
&          FaceBndryConnectionCounts ,
&          FaceBndryConnectionElems ,
&          FaceBndryConnectionZones)
    INTEGER*4      FaceNodeCounts (*)
    INTEGER*4      FaceNodes (*)
    INTEGER*4      FaceLeftElems (*)
    INTEGER*4      FaceRightElems (*)
    INTEGER*4      FaceBndryConnectionCounts (*)
    INTEGER*4      FaceBndryConnectionElems (*)
    INTEGER*2      FaceBndryConnectionZones (*)

```

C Syntax:

```

include TECIO.h
INTEGER4  TECPOLY111 (INTEGER4      *FaceNodeCounts ,
                        INTEGER4      *FaceNodes ,
                        INTEGER4      *FaceLeftElems ,
                        INTEGER4      *FaceRightElems ,
                        INTEGER4      *FaceBndryConnectionCounts ,
                        INTEGER4      *FaceBndryConnectionElems ,
                        INTEGER2      *FaceBndryConnectionZones) ;

```

Return Value:

0 if successful, -1 if unsuccessful.



Parameters:

Parameter	Description
FaceNodeCounts	An array used to define the number of nodes in each face. The array is dimensioned by the number of faces (defined in TECZNE111). This is NULL for polygonal zones, as each face in a polygonal zone has exactly two nodes.
FaceNodes	An array used to specify which nodes belong to which face. The array is dimensioned by TotalNumFaceNodes (defined in TECZNE111).
FaceLeftElems	An array used to define the left neighboring element for each face. The array is dimensioned by NumFaces (defined in TECZNE111).
FaceRightElems	An array used to define the right neighboring element for each face. The array is dimensioned by NumFaces (defined in TECZNE111).
FaceBndryConnection-Counts	An array used to define the number of boundary connections for each boundary face. The array is dimensioned by TotalNumBndryFaces (defined in TECZNE111).
FaceBndryConnectionElems	An array used to define the boundary element(s) to which each boundary face is connected. The array is dimensioned by TotalNumBndryConnections (defined in TECZNE111).
FaceBndryConnectionZones	An array used to define the zone(s) to which each boundary element belongs. The array is dimensioned by TotalNumBndryConnections (defined in TECZNE111).

Examples

Refer to the following sections for examples using **TECPOLY111**:

- [Section 2- 9.2 “Polygonal Example”](#)
- [Section 2- 9.3 “Multiple Polyhedral Zones”](#)
- [Section 2- 9.4 “Multiple Polygonal Zones”](#)
- [Section 2- 9.5 “Polyhedral Example”](#)



Adds a text box to the file.

FORTRAN Syntax:

```

    INTEGER*4 FUNCTION TECTXT111 (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	<i>Angle</i>
INTEGER*4	<i>Anchor</i>
DOUBLE PRECISION	<i>LineSpacing</i>
INTEGER*4	<i>TextColor</i>
INTEGER*4	<i>Scope</i>
INTEGER*4	<i>Clipping</i>
CHARACTER* (*)	<i>Text</i>
CHARACTER* (*)	<i>MFC</i>

C Syntax:

```
#include TECIO.h
INTEGER4 TECTXT111 (double      *XOrThetaPos,
                        double    *YOrRPosPos,
                        double    *ZOrUnusedPos,
                        INTEGER4  *PosCoordMode,
                        INTEGER4  *AttachToZone,
                        INTEGER4  *Zone,
                        INTEGER4  *Font,
                        INTEGER4  *FontHeightUnits,
                        double     *FontHeight,
                        INTEGER4  *BoxType,
                        double     *BoxMargin,
                        double     *BoxLineThickness,
                        INTEGER4  *BoxColor,
                        INTEGER4  *BoxFillColor,
                        double     *Angle,
                        INTEGER4  *Anchor,
                        double     *LineSpacing,
                        INTEGER4  *TextColor,
                        INTEGER4  *Scope,
                        INTEGER4  *Clipping,
                        char       *Text,
                        char       *MFC)
```

Return Value:

0 if successful, -1 if unsuccessful.



Parameters:

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.
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 6=Times Italic 1=Helvetica Bold 7=Times Bold 2=Greek 8=Times Italic Bold 3=Math 9=Courier 4=User-Defined 10=Courier Bold 5=Times
FontHeightUnits	Pointer to integer value specifying the font height units. 0=Grid 2=Point 1=Frame
FontHeight	Pointer to double value specifying the font height. If PosCoordMode is set to FRAME, the value range is zero to 100.
BoxType	Pointer to integer value specifying the box type. 0=None 2=Hollow 1=Filled
BoxMargin	Pointer to double value specifying the box margin (in frame units ranging from 0 to 100).



Examples

Refer to [Section 2- 9.8 “Text Example”](#) for an example of working with **TECTXT111**.

TECUSR111

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.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECUSR111(S)
CHARACTER*(*) S
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECUSR111(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.

TECVAUXSTR111

Writes an auxiliary data item to the data file for the specified variable. Must be called after [TECINI111](#) and before [TECEND111](#). 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 “Variable” from the “Show Auxiliary Data” menu and selecting the corresponding variable number from the menu.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECVAUXSTR111(Var, Name, Value)
```



INTEGER*4 *Var*
CHARACTER* (*) *Name*
CHARACTER* (*) *Value*

C Syntax:

```
#include TECIO.h
INTEGER4 TECVAUXSTR111 (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)";

TECVAUXSTR111 (&Var,
               &PressureUnitsName[0],
               &PressureUnitsValue[0]);
```



TECZAUXSTR111

Writes an auxiliary data item for the current zone to the data file. Must be called immediately after [TECZNE111](#) for the desired zone. Auxiliary data may be used by text, macros, equations (if it is numeric) and add-ons. It may be viewed directly in the *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.

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECZAUXSTR111 (Name, Value)
CHARACTER* (*)      Name
CHARACTER* (*)      Value
```

C Syntax:

```
#include TECIO.h
INTEGER4 TECZAUXSTR111 (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: **TECZAUXSTR111** must be called immediately after [TECZNE111](#) for the desired zone.

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



```
TECZAUSTR111 (&CreatedByName [0] ,
              &CreatedByValue [0] ) ;
```

TECZNE111

Writes header information about the next zone to be added to the data file. After **TECZNE111** is called, you must call [TECDAT111](#) one or more times. If the zone is a finite-element zone, call [TECNOD111](#) (cell-based zones) or [TECPOLY111](#) (face-based zones) after calling [TECDAT111](#).



[ZoneType](#), please note that some features in Tecplot 360 are limited by zone type. For example, iso-surfaces and slices are available for 3D zones types only (FETETRAHEDRON, FEBRICK, FEPOLYHEDRON and ORDERED - with K greater than 1).

However, the plot type that you specify (in Tecplot 360 once you have loaded your data) is not limited by your zone type. You may have a 3D zone displayed in a 2D Cartesian plot (and visa versa).

FORTRAN Syntax:

```
INTEGER*4 FUNCTION TECZNE111 (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    TECZNE111 (char
                INTEGER4    *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
IMxOrNumPts	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.
JMxOrNumElements	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.
KMxOrNumFaces	ORDERED FEPOLYGON FEPOLYHEDRON	For ordered zones, the number of nodes in the K index direction. For polyhedral and polygonal finite-element zones, it is the number of faces. Not used all other finite-element zone types.
ICellMax	N/A	Reserved for future use. Set to zero.



Parameter	Applies to Zone Type(s)	Notes
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 Section 8 - 2 “Time Aware” in the User’s Manual for additional information on working with transient data.
StrandID	ALL	<p>Scalar integer value specifying the strand to which the zone is associated.</p> <p>0 = static zone, not associated with a strand. Values greater than 0 indicate a zone is assigned to a given strand.</p> <p>Refer to Section 8 - 2 “Time Aware” in the User’s Manual for additional information on strands.</p> <p>NOTE: If you are converting your function calls from function calls 109 or older, use “0” for StrandID.</p>
ParentZone	ALL	<p>Scalar integer value representing the relationship between this zone and its parent. With a parent zone association, Tecplot 360 can generate a surface streamtrace on a no-slip boundary zone. A zone may not specify itself as its own parent.</p> <p>0 = indicates that this zone is not associated with a parent zone. >0 = A value greater than zero is considered this zone's parent.</p> <p>Refer to Section 8 - 2 “Time Aware” in the User’s Manual for additional information on parent zones and Section 16 - 3 “Surface streamtraces on no-slip boundaries” in the User’s Manual for additional information regarding no-slip boundaries.</p>
IsBlock	ALL	<p>Indicates whether the data will be passed into TECDAT111 in BLOCK or POINT format.</p> <p>0=POINT 1=BLOCK.</p>



Parameter	Applies to Zone Type(s)	Notes
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 TECFACE111 .
FaceNeighborMode	ORDERED FELINESEG FETRIANGLE FEQUADRILATERAL FETETRAHEDRON FEBRICK	Used for cell-based ^a finite-element and ordered zones only. The type of face connections that will be passed in routine TECFACE111 . 0=LocalOneToOne 2=GlobalOneToOne 1=LocalOneToMany 3=GlobalOneToMany
TotalNumFaceNodes	FEPOLYGON FEPOLYHEDRON	Used for face-based ^b finite-element zones. Total number of nodes for all faces. It is also the sum of the FaceNodeCounts array (defined in TECPOLY111). For polygonal zones this value is equivalent to 2 * NumFaces. NumFaces = the number of faces in the zone. Refer to Section 2-8.2 “FaceNodeCounts and FaceNodes” for simple example.
NumConnected-BoundaryFaces	FEPOLYGON FEPOLYHEDRON	Used for face-based ^b 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 Section 2- 8.1 “Boundary Faces and Boundary Connections” for simple example.
TotalNumBoundaryConnections	FEPOLYGON FEPOLYHEDRON	Used for face-based ^b 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 NumConnectedBoundaryFaces . Refer to Section 2- 8.1 “Boundary Faces and Boundary Connections” for simple example.



Parameter	Applies to Zone Type(s)	Notes
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. Refer to “Passive Variables” on page 7 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.
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 TECDAT111 . 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).
ShareConnectivity-FromZone	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.

- a. Cell-based finite-element zones: FELINESEG, FETRIANGLE, FEQUADRILATERAL, FETETRAHEDRON, and FEBRICK.
- b. Face-based finite-element zones: FEPOLYGON and FEPOLYHEDRON.

Examples:

Refer to the following examples for illustrations of working with **TECZNE111**:

- [Section 2- 9.1 “Face Neighbors”](#)
- [Section 2- 9.2 “Polygonal Example”](#)



- [Section 2- 9.3 “Multiple Polyhedral Zones”](#)
- [Section 2- 9.4 “Multiple Polygonal Zones”](#)
- [Section 2- 9.5 “Polyhedral Example”](#)
- [Section 2- 9.6 “IJ-ordered zone”](#)
- [Section 2- 9.7 “Switching between two files”](#)

2 - 8 Defining Polyhedral and Polygonal Data

Polyhedral data is defined using both **TECPOLY111** and **TECZNE111**. Via **TECZNE111** the number of nodes, faces, elements, boundary faces, and boundary connections are specified. **TECPOLY111** is used to specify the face mapping connections for the zone.

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.

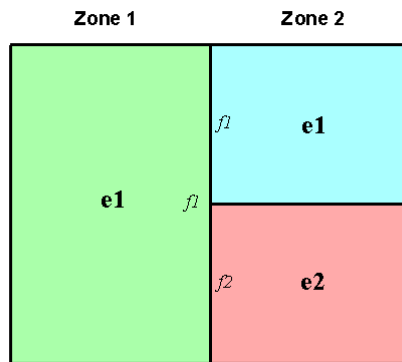
The remainder of this section provides simple examples illustrating how to define each of the parameters of **TECPOLY111**.

2- 8.1 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.
 - NumConnectedBndryFaces = 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.
 - NumConnectedBndryFaces = 2
 - TotalNumBndryConnections = 2



2- 8.2 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.

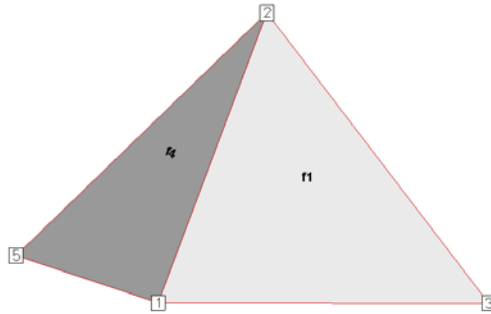


Figure 2-2. A simple pyramid. The remaining triangular faces are Faces 2 and 3. The bottom rectangular face is Face 5. Node 4 is obscured from view.

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 **TECZNE111**). The total number of face nodes is defined as:

$$\sum_{f=1}^F \text{NumNodesInFace}_f$$



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.

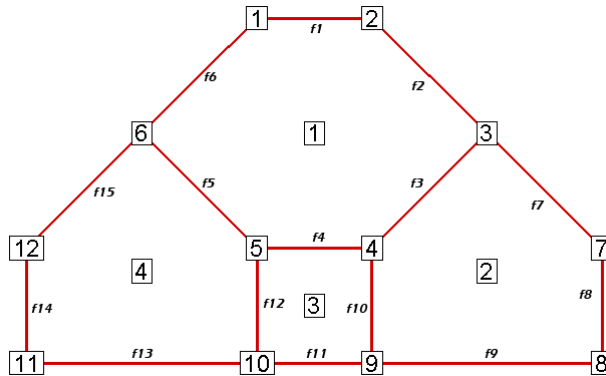
Consider the pyramid used above. Using the FaceNodeCounts array we have already defined and the figure, we can create the FaceNodes array for the pyramid.

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



2- 8.3 FaceRightElems and FaceLeftElems

After specifying the face map data (via 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 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”).



2- 8.4 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.

2 - 9 Examples

The following 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 [Section 2 - 6 “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 [Section 2 - 7 “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 [Section 2 - 4 “Binary Data File Function Calling Sequence”](#) for details.

2- 9.1 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 and malloc.h¹.

1. You may notice that malloc is used throughout the example. This is done to clearly indicate the dimensions required for each array. It is not required in practice.



For complete details on the parameters used and the function syntax for each TecIO function, refer to [Section 2 - 7 “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 [Section 2 - 4 “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).

```

Line 1  INTEGER4   Debug, I,  VIsDouble,  FileType, NumVars;
Line 2
Line 3  Debug      = 1;
Line 4  VIsDouble = 0;
Line 5  FileType  = 0;
Line 6  NumVars   = 3;
Line 7
Line 8  I = TECINI111("Face Neighbors Example",
Line 9                      "X Y P",
Line 10                     "FaceNeighbors.plt",
Line 11                     ".",
Line 12                     &FileType,
Line 13                     &Debug,
Line 14                     &VIsDouble);

```

- [Line 8](#) - Specifies the name of the entire data set. When the file is loaded into Tecplot, the value is available via the **Data Set Information** dialog.
- [Line 9](#) - 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 Variable 1). When referring to variables in other TecIO functions, you will refer to the variable by its number.

Step 2: Create Zone 1

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

```

Line 15 double      SolTime;
Line 16 INTEGER4   ZoneType, NumPts,      NumElems,
Line 17           NumFaces, StrandID,  ParentZn, IsBlock,
Line 18           ICellMax, JCellMax, KCellMax, NFConns,

```



```
Line 19          FNMode,   ShrConn,
Line 20          TotalNumFaceNodes,
Line 21          NumConnectedBoundaryFaces,
Line 22          TotalNumBoundaryConnections;
Line 23 ZoneType = 3; // set the zone type to FEQuadilateral
Line 24 NumPts   = 6;
Line 25 NumElems = 2;
Line 26 NumFaces = 8;
Line 27 ICellMax = JCellMax = KCellMax = 0; //not used.
Line 28 SolTime  = 360.0;
Line 29 StrandID = 0;
Line 30 ParentZn = 0;
Line 31 IsBlock  = 1;
Line 32 NFConns  = 1;
Line 33 FNMode   = 2;
Line 34 TotalNumFaceNodes = 1; // not used for FEQuad zones
Line 35 NumConnectedBoundaryFaces = 1; // not used for
Line 36                               // FEQuad zones
Line 37 TotalNumBoundaryConnections = 1; // not used for
Line 38                               // FEQuad zones
Line 39 ShrConn          = 0;
Line 40 INTEGER4      ValueLocation[3] = {1,1,1};
Line 41 I = TECZNE111("Zone 1",
Line 42                &ZoneType,
Line 43                &NumPts,
Line 44                &NumElems,
Line 45                &NumFaces,
Line 46                &ICellMax,
Line 47                &JCellMax,
Line 48                &KCellMax,
Line 49                &SolTime,
Line 50                &StrandID,
Line 51                &ParentZn,
```




```

Line 52          &IsBlock,
Line 53          &NFConns,
Line 54          &FNMode,
Line 55          &TotalNumFaceNodes,
Line 56          &NumConnectedBoundaryFaces,
Line 57          &TotalNumBoundaryConnections,
Line 58          NULL,
Line 59          ValueLocation,
Line 60          NULL,
Line 61          &ShrConn);

```

- [Line 32](#) - 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.
- [Line 33](#) - Specify the Face Neighbor Mode. A value of 2 indicates 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 across zones; a value of “local” indicates that the face neighbor(s) is/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.
- [Line 40](#) - 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.

Step 3: Define the node numbering for Zone 1

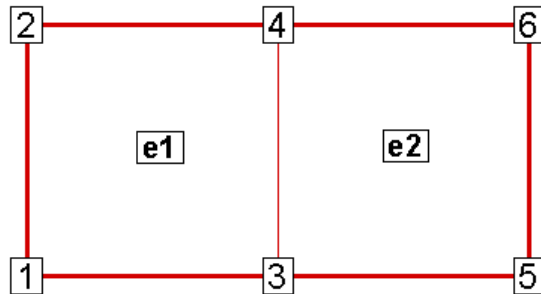
For this example, we will create 2 rectangular cells in Zone 1. Before defining your variables, you must establish a consistent node numbering scheme for your data. Once the node numbers are defined, supply the variable values in the node numbering order. In this example, Node 1 is defined at $X = 0$ 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 the [Section “TECZNE111”](#) on page 42 for details.

```
Line 62  float      *X, *Y, *P;
Line 63  X = (float*) malloc( NumPts * sizeof(float));
Line 64  Y = (float*) malloc( NumPts * sizeof(float));
Line 65  P = (float*) malloc( NumPts * sizeof(float));
Line 66
Line 67  X[0] = 0;
Line 68  X[1] = 0;
Line 69  X[2] = 1;
Line 70  X[3] = 1;
Line 71  X[4] = 2;
Line 72  X[5] = 2;
Line 73  Y[0] = 0;
Line 74  Y[1] = 1;
Line 75  Y[2] = 0;
Line 76  Y[3] = 1;
Line 77  Y[4] = 0;
Line 78  Y[5] = 1;
```



```

Line 79  /* we are using a pressure variable, but the values are not
Line 80  * important for the example.*/
Line 81  INTEGER4  ii;
Line 82  for(ii=0; ii < NumPts; ii++)
Line 83  P[ii] = float(NumPts - ii);
Line 84  INTEGER4  DIsDouble = 0;
Line 85
Line 86  /* Call TECDAT once for each variable */
Line 87  I   = TECDAT111 (&NumPts, &X[0], &DIsDouble) ;
Line 88  I   = TECDAT111 (&NumPts, &Y[0], &DIsDouble) ;
Line 89  I   = TECDAT111 (&NumPts, &P[0], &DIsDouble) ;

```

- [Line 84](#) - Set DIsDouble to zero to use variables in float format. Set the value to one to use double format.

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.

```

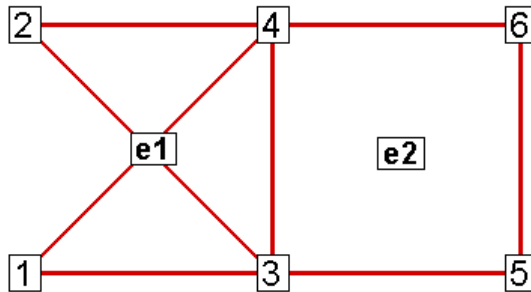
Line 90  INTEGER4 ConnList[8] = {1,3,4,2,
Line 91  3,5,6,4};
Line 92  I   = TECNOD111 (ConnList) ;

```





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 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.

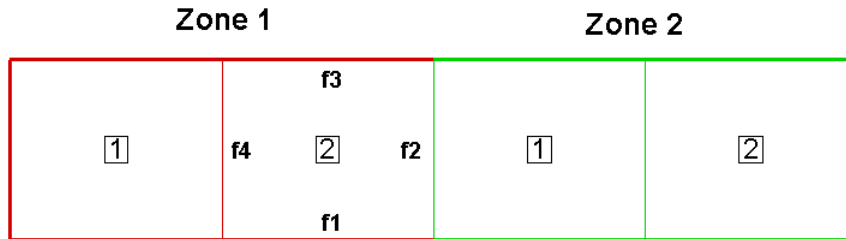
In this example, we are creating a face neighbor connection between Cell 2 in Zone 1 and Cell 1 in Zone 2. The information required when specifying face neighbors depends upon the type of connection. Refer to [Section “TECFACE111”](#) on page 19 for details.

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



The face numbering for cell-based finite-elements is defined using [Figure 2-1](#) on page 21. In this example, Face 2 in Cell 2 in the current zone is connected to Cell 1 in Zone 2.



```
Line 93  INTEGER4 FaceConn[4] = {2,2,2,1};
Line 94  I    = TECFACE111 (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.

```
Line 95  I = TECZNE111 ("Zone 2",
Line 96                                &ZoneType,
Line 97                                &NumPts,
Line 98                                &NumElems,
Line 99                                &NumFaces,
Line 100                               &ICellMax,
Line 101                               &JCellMax,
Line 102                               &KCellMax,
Line 103                               &SolTime,
Line 104                               &StrandID,
Line 105                               &ParentZn,
Line 106                               &IsBlock,
Line 107                               &NFConns,
```



```

Line 108          &FNMode,
Line 109          &TotalNumFaceNodes,
Line 110          &NumConnectedBoundaryFaces,
Line 111          &TotalNumBoundaryConnections,
Line 112          NULL,
Line 113          ValueLocation,
Line 114          NULL,
Line 115          &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).

```

Line 116 float      *X2, *P2;
Line 117 X2 = (float*) malloc( NumPts * sizeof(float));
Line 118 P2 = (float*) malloc( NumPts * sizeof(float));
Line 119
Line 120 for(ii=0; ii < NumPts; ii++)
Line 121 {
Line 122     X2[ii] = X[ii] + 2;
Line 123     P2[ii] = 2 * float(ii);
Line 124 }
Line 125 I   = TECDAT111 (&NumPts, &X2[0], &DIsDouble);
Line 126 I   = TECDAT111 (&NumPts, &Y[0], &DIsDouble);
Line 127 I   = TECDAT111 (&NumPts, &P2[0], &DIsDouble);
Line 128 free(X);
Line 129 free(Y);
Line 130 free(P);
Line 131 free(X2);
Line 132 free(P2);

```



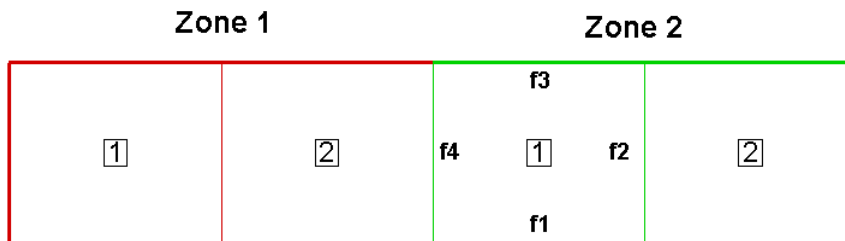
Step 9: Define the connectivity list for Zone 2

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.

```
Line 133 I = TECNOD111(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.



```
Line 134 INTEGER4 FaceConn2[4] = {1,4,1,2}; /* cell 1, face 4 in
Line 135                                     current zone is a
Line 136                                     neighbor to cell 2 in
Line 137                                     zone 1 */
Line 138 I = TECFACE111(FaceConn2);
```

Step 11: Close the file

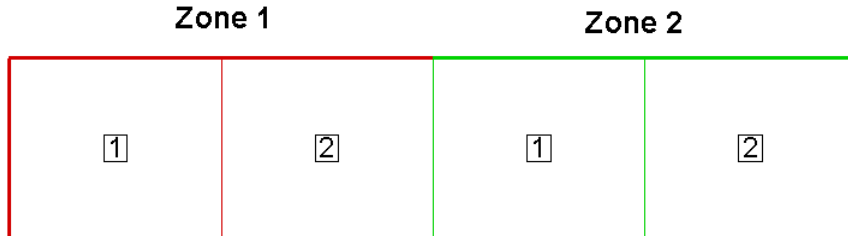
Call TECEND to close the file.

```
Line 139 I = TECEND111();
```

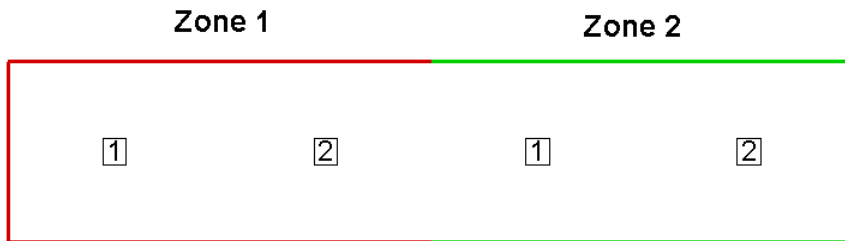


Summary

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:

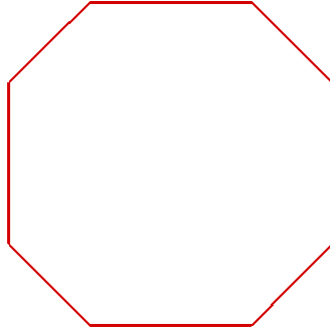


If we had not included face neighbor connections, an Edge line would be drawn in between the two zones.



2- 9.2 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` and `malloc.h`¹. The source files for this example are included in your Tecplot 360 installation under the `\util\tecio\polyhedral\octagon` sub-directory.

For complete details on the parameters used and the function syntax for each TecIO function, refer to [Section 2 - 7 “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 [Section 2 - 4 “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).

```

Line 140 INTEGER4   Debug, I,   VisDouble,   FileType, NumVars;
Line 141
Line 142 Debug      = 1;
Line 143 VisDouble = 0;
Line 144 FileType  = 0;
Line 145 NumVars   = 3;
Line 146

```

1. You may notice that `malloc` is used throughout the example. This is done to clearly indicate the dimensions required for each array. It is not required in practice.



```

Line 147 I = TECINI111("Octagon",
Line 148           "X Y P",
Line 149           "Octagon.plt",
Line 150           ".",
Line 151           &FileType,
Line 152           &Debug,
Line 153           &VIsDouble);

```

- [Line 147](#) - Specifies the name of the entire data set. When the file is loaded into Tecplot, the value is available via the **Data Set Information** dialog.
- [Line 148](#) - 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 Variable 1). When referring to variables in other TecIO functions, you will refer to the variable by its number.

Step 2: Create Zone 1

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

```

Line 154 double SolTime;
Line 155 INTEGER4 NumNodes, NumElems, NumFaces, ZoneType,
Line 156           StrandID, ParentZn, IsBlock, ICellMax, JCellMax,
Line 157           KCellMax, NFConns, FNMode, NumFaceNodes, NumBFaces,
Line 158           NumBConnections, ShrConn, ValueLocation[3];
Line 159 ZoneType = 6; /* FEPolygon */
Line 160 NumNodes = 8;
Line 161 NumElems = 1;
Line 162 NumFaces = 8;
Line 163 ICellMax = JCellMax = KCellMax = 0; //Not Used
Line 164 SolTime = 360.0;
Line 165 StrandID = 0; /* Static Zone */
Line 166 ParentZn = 0; /* No Parent */
Line 167 IsBlock = 1; /* Block */
Line 168 NFConns = 0;
Line 169 FNMode = 0;

```



```

Line 170 NumFaceNodes = 2 * NumNodes ;
Line 171 NumBFaces      = 0 ;
Line 172 NumBConnections = 0 ;
Line 173 ShrConn       = 0 ;
Line 174 ValueLocation[0] = 1 ;
Line 175 ValueLocation[1] = 1 ;
Line 176 ValueLocation[2] = 1 ;
Line 177 I = TECZNE111("Octagonal Zone",
Line 178             &ZoneType,
Line 179             &NumNodes,
Line 180             &NumElems,
Line 181             &NumFaces,
Line 182             &ICellMax,
Line 183             &JCellMax,
Line 184             &KCellMax,
Line 185             &SolTime,
Line 186             &StrandID,
Line 187             &ParentZn,
Line 188             &IsBlock,
Line 189             &NFConns,
Line 190             &FNMode,
Line 191             &NumFaceNodes,
Line 192             &NumBFaces,
Line 193             &NumBConnections,
Line 194             NULL,
Line 195             ValueLocation,
Line 196             NULL,
Line 197             &ShrConn) ;

```

- [Line 160](#), [Line 161](#) and [Line 162](#) - For this example, we are creating a single octagonal cell. As such, there are eight nodes and faces and one element.
- [Line 170](#) - 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.

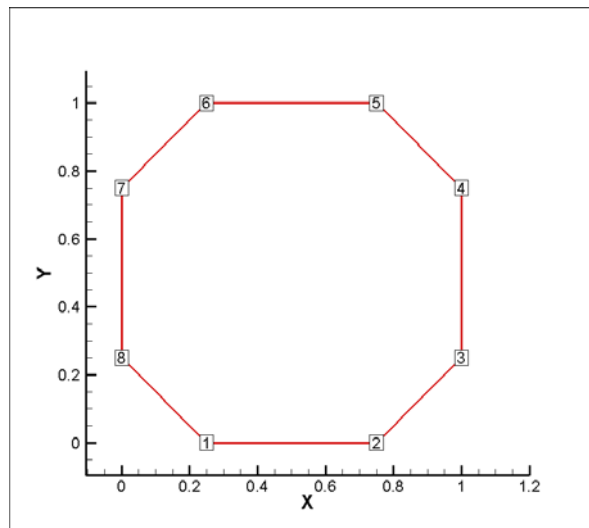


- [Line 195](#) - When ValueLocation is not specified, Tecplot will treat all variables as nodal variables. We are explicitly setting all variables to be nodal for illustration purposes only.

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 TECZNE - [Line 195](#)), 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 [Section "TECZNE111"](#) on page 42 for details.



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

$V1 = \{ValueAtNode1, ValueAtNode2, \dots, ValueAtNodeN\}$

where N is the total number of nodes

```
Line 198  float      *X, *Y, *P;
Line 199  X = (float*) malloc( NumNodes * sizeof(float ) );
Line 200  Y = (float*) malloc( NumNodes * sizeof(float ) );
Line 201  P = (float*) malloc( NumNodes * sizeof(float ) );

Line 202  X[0] = .25;
Line 203  X[1] = .75;
Line 204  X[2] = 1.0;
Line 205  X[3] = 1.0;
Line 206  X[4] = .75;
Line 207  X[5] = .25;
Line 208  X[6] = 0.0;
Line 209  X[7] = 0.0;

Line 210  Y[0] = 0.0;
Line 211  Y[1] = 0.0;
Line 212  Y[2] = .25;
Line 213  Y[3] = .75;
Line 214  Y[4] = 1.0;
Line 215  Y[5] = 1.0;
Line 216  Y[6] = .75;
Line 217  Y[7] = .25;

Line 218  /* we are adding a pressure variable, but the values
Line 219      are insignificant */
Line 220  INTEGER4 ii;
Line 221  for (ii = 0; ii < NumNodes; ii++)
Line 222      P[ii] = .5;
```



Step 5: Write out the field data using TECDAT

Now that the variables are defined, we can write them to the data file using TECDAT.

```

Line 223  INTEGER4      DIsDouble;
Line 224  DIsDouble = 0;
Line 225
Line 226  I    = TECDAT111 (&NumNodes,  &X[0],  &DIsDouble) ;
Line 227  I    = TECDAT111 (&NumNodes,  &Y[0],  &DIsDouble) ;
Line 228  I    = TECDAT111 (&NumNodes,  &P[0],  &DIsDouble) ;
Line 229
Line 230  free (X) ;
Line 231  free (Y) ;
Line 232  free (P) ;

```

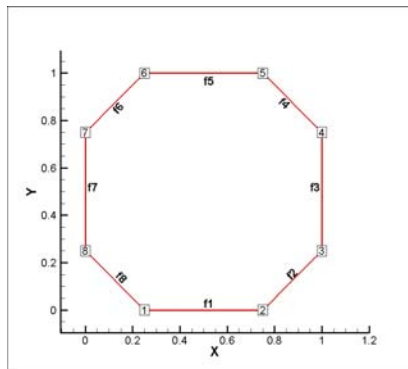
- [Line 224](#) - Set IsDouble to zero to use variables in float format. Set IsDouble to one to use variables in double format.

Step 6: 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 [Section 2- 9.4 “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.

```

Line 233  INTEGER4      *FaceNodes;
Line 234  FaceNodes=(INTEGER4*)malloc(NumFaceNodes *sizeof(INTEGER4));
Line 235
Line 236  FaceNodes[0] = 1;
Line 237  FaceNodes[15] = 1;
Line 238
Line 239  INTEGER4      jj;
Line 240  jj = 2;
Line 241  for (ii = 1; ii < 15; ii+=2)
Line 242  {
Line 243      FaceNodes[ii] = jj;
Line 244      FaceNodes[ii+1] = jj;
Line 245      jj++;
Line 246  }

```

Step 7: 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 [Section 2- 8.3 “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).

```

Line 247  INTEGER4      *FaceLeftElems, *FaceRightElems;
Line 248
Line 249  FaceLeftElems = (INTEGER4*)malloc(NumFaces *
sizeof(INTEGER4));
Line 250  FaceRightElems = (INTEGER4*)malloc(NumFaces*
sizeof(INTEGER4));

```



```
Line 251
Line 252   for (ii = 0; ii < NumFaces; ii++)
Line 253     {
Line 254         FaceLeftElems[ii]   = 0;
Line 255         FaceRightElems[ii]  = 1;
Line 256     }
Line 257 /* Write the polyhedral data to the file. */
Line 258
Line 259   I = TECPOLY111(NULL,
Line 260                 &FaceNodes[0],
Line 261                 &FaceLeftElems[0],
Line 262                 &FaceRightElems[0],
Line 263                 NULL,
Line 264                 NULL,
Line 265                 NULL);
Line 266
Line 267   free (FaceNodes);
Line 268   free (FaceLeftElems);
Line 269   free (FaceRightElems);
```

Step 8: Close the file

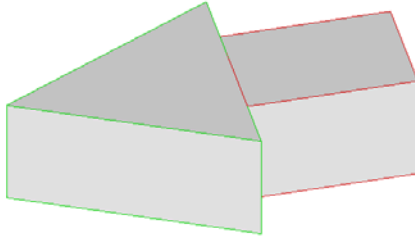
Call TECEND to close the file.

```
Line 270   I = TECEND111 ();
```



2- 9.3 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` and `malloc.h`¹. The source files for this example are included in your Tecplot 360 installation under the `\util\tecio\polyhedral\arrow` subdirectory.

For complete details on the parameters used and the function syntax for each TecIO function, refer to [Section 2 - 7 “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 [Section 2 - 4 “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).

```

Line 1  INTEGER4  Debug, I, VisDouble, FileType;
Line 2
Line 3  Debug      = 1;
Line 4  VisDouble = 0;
Line 5  FileType  = 0;
Line 6
Line 7  I = TECINI111("Example: Multiple polyhedral zones",

```

1. You may notice that `malloc` is used throughout the example. This is done to clearly indicate the dimensions required for each array. It is not required in practice.



```

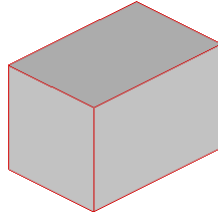
Line 8           "X Y Z P",
Line 9           "Arrow.plt",
Line 10          ".",
Line 11          &FileType,
Line 12          &Debug,
Line 13          &VIsDouble);

```

- [Line 7](#) - Specifies the name of the entire data set. When the file is loaded into Tecplot, the value is available via the **Data Set Information** dialog.
- [Line 8](#) - 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 Variable 1). When referring to variables in other TecIO functions, you will refer to the variable by its number.

Step 2: Create Zone 1 (rectangle)

After TECINI is called, call TECZNE 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.



```

Line 14  double   SolutionTime;
Line 15  INTEGER4 ZoneType, NumPts_Rect, NumElems_Rect,
Line 16  NumFaces_Rect, ICellMax, JCellMax, KCellMax,
Line 17  StrandID, ParentZone, IsBlock, NumFaceConnections,
Line 18  FaceNeighborMode, ValueLocation[4], SharConn,
Line 19  TotalNumFaceNodes_Rect, TotalNumBndryConns_Rect
Line 20  NumConnectedBndryFaces_Rect;
Line 21  //TECZNE Parameters

```



```
Line 22  ZoneType                = 7;
Line 23  NumPts_Rect            = 8;
Line 24  NumElems_Rect         = 1;
Line 25  NumFaces_Rect         = 6;
Line 26  ICellMax = JCellMax = KCellMax = 0;
Line 27  SolutionTime          = 0.0;
Line 28  StrandID              = 0;
Line 29  ParentZone            = 0;
Line 30  IsBlock               = 1;
Line 31  NumFaceConnections     = 0; /* ...not used */
Line 32  FaceNeighborMode      = 1; /* ...not used */
Line 33  SharConn               = 0;
Line 34  TotalNumFaceNodes_Rect = 24;
Line 35  NumConnectedBndryFaces_Rect = 1;
Line 36  TotalNumBndryConns_Rect = 1;
Line 37  ValueLocation[0]      = 1;
Line 38  ValueLocation[1]      = 1;
Line 39  ValueLocation[2]      = 1;
Line 40  ValueLocation[3]      = 0;
Line 41  I = TECZNE111("Zone 1: Rectangular Solid",
Line 42          &ZoneType,
Line 43          &NumPts_Rect,
Line 44          &NumElems_Rect,
Line 45          &NumFaces_Rect,
Line 46          &ICellMax,
Line 47          &JCellMax,
Line 48          &KCellMax,
Line 49          &SolutionTime,
Line 50          &StrandID,
Line 51          &ParentZone,
Line 52          &IsBlock,
Line 53          &NumFaceConnections,
Line 54          &FaceNeighborMode,
```



<i>Line 55</i>	<code>&TotalNumFaceNodes_Rect,</code>
<i>Line 56</i>	<code>&NumConnectedBndryFaces_Rect,</code>
<i>Line 57</i>	<code>&TotalNumBndryConns_Rect,</code>
<i>Line 58</i>	<code>NULL,</code>
<i>Line 59</i>	<code>ValueLocation,</code>
<i>Line 60</i>	<code>NULL,</code>
<i>Line 61</i>	<code>&SharConn) ;</code>

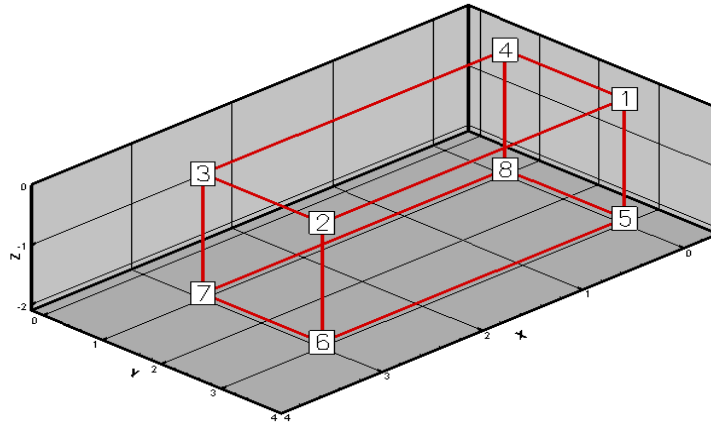
- [Line 22](#) - Set the zone type to polyhedral.
- [Line 34](#) - In a rectangular solid, each face is composed of four nodes. As such, the total number of face nodes is twenty-four (four nodes for each of the six faces).
- [Line 35](#) - There is one connected boundary face in this zone (the face on the rectangle adjacent to the arrowhead). Refer to [Section 2- 8.1 “Boundary Faces and Boundary Connections”](#) for additional information.
- [Line 36](#) - 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. Refer to [Section 2- 8.1 “Boundary Faces and Boundary Connections”](#) for additional information.
- [Line 40](#) - 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).

Step 3: Set variable values for Zone 1 (rectangle)

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 rectangle, the values are defined explicitly for simplicity.



Using the picture below, define the variable values.



For nodal variables, provides 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 via TECZNE.

```

Line 62 //set variable values (X_Rect, Y_Rect, Z_Rect & P_Rect)
Line 63
Line 64 INTEGER4 ii;
Line 65 double *X_Rect, *Y_Rect, *Z_Rect, *P_Rect;
Line 66
Line 67 X_Rect = (double*)malloc( NumPts_Rect * sizeof(double) );
Line 68 Y_Rect = (double*)malloc( NumPts_Rect * sizeof(double) );
Line 69 Z_Rect = (double*)malloc( NumPts_Rect * sizeof(double) );
Line 70 P_Rect = (double*)malloc( NumElems_Rect * sizeof(double) );
Line 71
Line 72 for(ii = 0; ii <= NumPts_Rect/2; ii+= 4)
Line 73 {
Line 74 X_Rect[ii] = 0;
Line 75 X_Rect[ii+1] = 3;
Line 76 X_Rect[ii+2] = 3;
Line 77 X_Rect[ii+3] = 0;

```



```
Line 78
Line 79     Y_Rect[ii]   = 3;
Line 80     Y_Rect[ii+1] = 3;
Line 81     Y_Rect[ii+2] = 1;
Line 82     Y_Rect[ii+3] = 1;
Line 83     }
Line 84
Line 85     for(ii = 0; ii<4; ii++)
Line 86         Z_Rect[ii] = 0;
Line 87
Line 88     for(ii = 4; ii < NumPts_Rect; ii++)
Line 89         Z_Rect[ii] = -2;
Line 90
Line 91     P_Rect[0] = 10;
Line 92
Line 93     INTEGER4 IsDouble = 1;
Line 94     I = TECDAT111 (&NumPts_Rect,   X_Rect, &IsDouble);
Line 95     I = TECDAT111 (&NumPts_Rect,   Y_Rect, &IsDouble);
Line 96     I = TECDAT111 (&NumPts_Rect,   Z_Rect, &IsDouble);
Line 97     I = TECDAT111 (&NumElems_Rect, P_Rect, &IsDouble);
```

- [Line 93](#) - Specify that the variable values are in double format. Set IsDouble to 0 to use variables in float format.
- [Line 97](#) - Because we specified that the pressure variable is cell-centered in [Line 40](#), we need to provide NumElems number of values of data for Pressure.



Step 4: Define the facemap data for Zone 1

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

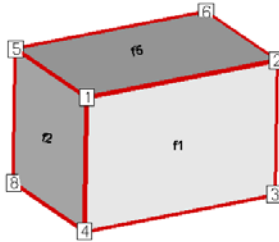


Figure 2-3. Zone 2 of the sample data. Node 7 is obscured from view and located in the back-left hand corner. Face 6 is the bottom face. Face 3 is opposite Face 1 and Face 4 is opposite Face 2.

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 [Section 2- 8.2 “FaceNodeCounts and FaceNodes”](#) for additional information.

```

Line 98  INTEGER4 *FaceNodeCounts_Rect = (INTEGER4*)malloc(NumFaces_Rect *
        sizeof(INTEGER4));
Line 99
Line 100 INTEGER4 *FaceNodes_Rect = (INTEGER4*)malloc(TotalNumFaceNodes_Rect
        * sizeof(INTEGER4));
Line 101
Line 102  //For this particular zone, each face has 4 nodes
Line 103  for(ii=0; ii<NumFaces_Rect; ii++)
Line 104      FaceNodeCounts_Rect[ii] = 4;
Line 105
Line 106  //Nodes for Face 1
Line 107  FaceNodes_Rect[0]   = 1;
Line 108  FaceNodes_Rect[1]   = 2;
Line 109  FaceNodes_Rect[2]   = 3;
Line 110  FaceNodes_Rect[3]   = 4;
Line 111
Line 112  //Nodes for Face 2

```



```
Line 113  FaceNodes_Rect[4]  = 1;
Line 114  FaceNodes_Rect[5]  = 4;
Line 115  FaceNodes_Rect[6]  = 8;
Line 116  FaceNodes_Rect[7]  = 5;
Line 117
Line 118  //Nodes for Face 3
Line 119  FaceNodes_Rect[8]  = 5;
Line 120  FaceNodes_Rect[9]  = 8;
Line 121  FaceNodes_Rect[10] = 7;
Line 122  FaceNodes_Rect[11] = 6;
Line 123
Line 124  //Nodes for Face 4
Line 125  FaceNodes_Rect[12] = 2;
Line 126  FaceNodes_Rect[13] = 6;
Line 127  FaceNodes_Rect[14] = 7;
Line 128  FaceNodes_Rect[15] = 3;
Line 129
Line 130  //Nodes for Face 5
Line 131  FaceNodes_Rect[16] = 6;
Line 132  FaceNodes_Rect[17] = 2;
Line 133  FaceNodes_Rect[18] = 1;
Line 134  FaceNodes_Rect[19] = 5;
Line 135
Line 136  //Nodes for Face 6
Line 137  FaceNodes_Rect[20] = 3;
Line 138  FaceNodes_Rect[21] = 7;
Line 139  FaceNodes_Rect[22] = 8;
Line 140  FaceNodes_Rect[23] = 4;
```

- [Line 98](#) - 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.



- [Line 100](#) - 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.



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 [Section 2- 8.3 “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 [Line 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).

```

Line 141  INTEGER4 *FaceLeftElems_Rect = (INTEGER4*)malloc(NumFaces_Rect *
          sizeof(INTEGER4));
Line 142
Line 143  INTEGER4 *FaceRightElems_Rect = (INTEGER4*)malloc(NumFaces_Rect *
          sizeof(INTEGER4));
Line 144
Line 145  for(ii=0;ii<NumFaces_Rect;ii++)
Line 146  {

```



```

Line 147     FaceRightElems_Rect[ii] = 1;
Line 148     FaceLeftElems_Rect[ii]  = 0;
Line 149     }
Line 150
Line 151     FaceLeftElems_Rect[3]    = -1;

```

- [Line 151](#) - The negative value in the FaceLeftElems array indicates that the face is connected to an element in another zone. In this case, Face 4¹ 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 5².

Step 6: Define boundary connections for Zone 1

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

```

Line 152  INTEGER4 *FaceBndryConnectionCounts_Rect =
          (INTEGER4*)malloc(NumConnectedBndryFaces_Rect * sizeof(INTEGER4));
Line 153
Line 154  INTEGER4 *FaceBndryConnectionElems_Rect =
          (INTEGER4*)malloc(TotalNumBndryConns_Rect * sizeof(INTEGER4));
Line 155
Line 156  INTEGER2 *FaceBndryConnectionZones_Rect =
          (INTEGER2*)malloc(TotalNumBndryConns_Rect * sizeof(INTEGER2));
Line 157  FaceBndryConnectionCounts_Rect[0] = 1;
Line 158  FaceBndryConnectionElems_Rect[0]  = 1;

```

1. In C, the index values are zero-based. However, Tecplot uses a 1-based numbering scheme for nodes, faces, elements and zones.
2. This is only valid if the neighboring elements are one-to-one. If the neighboring elements are one-to-many, you will need to refer to the FaceBndryConnectionCounts array to determine the position in the FaceBoundaryConnectionElems and Zones arrays.



```

Line 159  FaceBndryConnectionZones_Rect[0] = 2;
Line 160
Line 161  I = TECPOLY111(FaceNodeCounts_Rect,
Line 162                      FaceNodes_Rect,
Line 163                      FaceLeftElems_Rect,
Line 164                      FaceRightElems_Rect,
Line 165                      FaceBndryConnectionCounts_Rect,
Line 166                      FaceBndryConnectionElems_Rect,
Line 167                      FaceBndryConnectionZones_Rect);
Line 168
Line 169  /* cleanup */
Line 170  free(X_Rect);
Line 171  free(Y_Rect);
Line 172  free(Z_Rect);
Line 173  free(P_Rect);
Line 174  free(FaceNodeCounts_Rect);
Line 175  free(FaceNodes_Rect);
Line 176  free(FaceLeftElems_Rect);
Line 177  free(FaceRightElems_Rect);
Line 178  free(FaceBndryConnectionCounts_Rect);
Line 179  free(FaceBndryConnectionElems_Rect);
Line 180  free(FaceBndryConnectionZones_Rect);

```

- [Line 157](#) - 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].

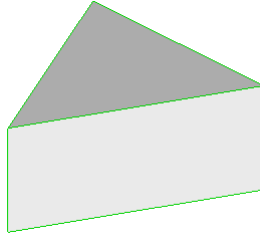
In this example, the total number of connected boundary faces (specified via TECZNE in [Line 35](#)) is equal to one, so the FaceBoundaryConnectionCounts array contains a single value (1).

- [Line 158](#) and [Line 159](#) - 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 boundary connection face is connected to Element 1 in Zone 2.



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 [Line 2](#) that are not relevant to this tutorial.



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

```

Line 181 INTEGER4   NumPts_Tri, NumElems_Tri, NumFaces_Tri,
Line 182           TotalNumFaceNodes_Tri, NumConnectedBndryFaces_Tri,
Line 183           TotalNumBndryConns_Tri;
Line 184
Line 185 //TECZNE Parameters
Line 186   NumPts_Tri           = 6;
Line 187   NumElems_Tri        = 1;
Line 188   NumFaces_Tri        = 5;
Line 189   TotalNumFaceNodes_Tri = 18;
Line 190   NumConnectedBndryFaces_Tri = 1;
Line 191   TotalNumBndryConns_Tri = 2;
Line 192
Line 193   I = TECZNE111("Zone 2: Prism",
Line 194               &ZoneType,
Line 195               &NumPts_Tri,
Line 196               &NumElems_Tri,
Line 197               &NumFaces_Tri,
Line 198               &ICellMax,
Line 199               &JCellMax,
Line 200               &KCellMax,

```



```

Line 201          &SolutionTime,
Line 202          &StrandID,
Line 203          &ParentZone,
Line 204          &IsBlock,
Line 205          &NumFaceConnections,
Line 206          &FaceNeighborMode,
Line 207          &TotalNumFaceNodes_Tri,
Line 208          &NumConnectedBndryFaces_Tri,
Line 209          &TotalNumBndryConns_Tri,
Line 210          NULL,
Line 211          ValueLocation,
Line 212          NULL,
Line 213          &SharConn) ;

```

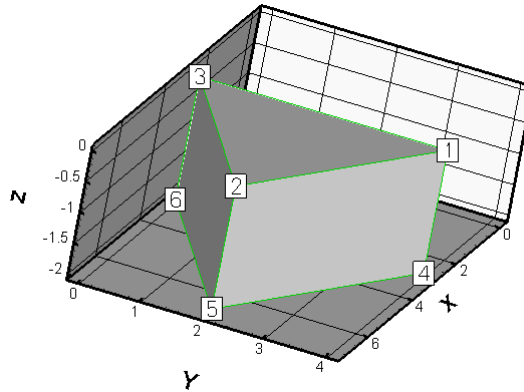
- [Line 189](#) - The prism is composed of two triangular faces and three rectangular faces. The total number of face nodes is the sum of the nodes in each triangular face (2 times 3) and the nodes in each rectangular face (3 times 4).
- [Line 190](#) - As in Zone 1, Zone 2 has one connected boundary face, the face that is connected to Zone 1.
- [Line 191](#) - 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.

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.



```

Line 214  double   *X_Tri, *Y_Tri, *Z_Tri, *P_Tri;
Line 215
Line 216  X_Tri = (double*)malloc(NumPts_Tri * sizeof(double));
Line 217  Y_Tri = (double*)malloc(NumPts_Tri * sizeof(double));
Line 218  Z_Tri = (double*)malloc(NumPts_Tri * sizeof(double));
Line 219  P_Tri = (double*)malloc(NumElems_Tri * sizeof(double));
Line 220
Line 221  for(ii = 0; ii<= NumPts_Tri/2; ii+= 3)
Line 222  {
Line 223      X_Tri[ii]   = 3;
Line 224      X_Tri[ii+1] = 7;
Line 225      X_Tri[ii+2] = 3;
Line 226
Line 227      Y_Tri[ii]   = 4;
Line 228      Y_Tri[ii+1] = 2;
Line 229      Y_Tri[ii+2] = 0;
Line 230  }
Line 231
Line 232  for(ii = 0; ii<3; ii++)

```



```

Line 233   Z_Tri[ii] = 0;
Line 234
Line 235   for(ii = 3; ii < NumPts_Tri; ii++)
Line 236     Z_Tri[ii] = -2;
Line 237
Line 238   P_Tri[0] = 20;
Line 239
Line 240   I = TECDAT111(&NumPts_Tri, X_Tri, &IsDouble);
Line 241   I = TECDAT111(&NumPts_Tri, Y_Tri, &IsDouble);
Line 242   I = TECDAT111(&NumPts_Tri, Z_Tri, &IsDouble);
Line 243   I = TECDAT111(&NumElems_Tri,P_Tri, &IsDouble);

```

- [Line 243](#) - In [Line 40](#), we specified that the variable 4 (pressure) is cell-centered. As such, only NumElems number of values needs to be written to the data file for the pressure variable.

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.

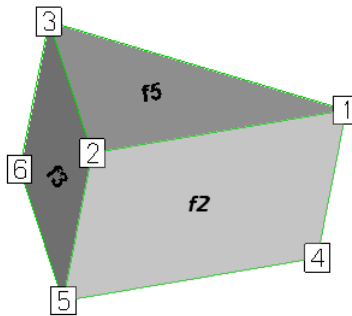


Figure 2-4. The arrowhead with three faces visible (Face 2, Face 3 and Face 5). The remaining rectangular face is Face 1, and the remaining triangular face is Face 4).

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].

```
Line 244  INTEGER4      *FaceNodeCounts_Tri,  *FaceNodes_Tri;
Line 245
Line 246  FaceNodeCounts_Tri = (INTEGER4*)malloc(NumFaces_Tri *
        sizeof(INTEGER4));
Line 247
Line 248  FaceNodes_Tri = (INTEGER4*)malloc(TotalNumFaceNodes_Tri *
        sizeof(INTEGER4));
Line 249
Line 250  for(ii=0;ii<3;ii++)
Line 251      FaceNodeCounts_Tri[ii] = 4;
Line 252
Line 253  for(ii=3;ii<NumFaces_Tri;ii++)
Line 254      FaceNodeCounts_Tri[ii] = 3;
Line 255
Line 256  //Nodes for Face 1
Line 257  FaceNodes_Tri[0]  = 1;
Line 258  FaceNodes_Tri[1]  = 3;
Line 259  FaceNodes_Tri[2]  = 6;
Line 260  FaceNodes_Tri[3]  = 4;
Line 261
Line 262  //Nodes for Face 2
Line 263  FaceNodes_Tri[4]  = 1;
Line 264  FaceNodes_Tri[5]  = 4;
Line 265  FaceNodes_Tri[6]  = 5;
Line 266  FaceNodes_Tri[7]  = 2;
Line 267
Line 268  //Nodes for Face 3
```




```

Line 269  FaceNodes_Tri[8]   = 3;
Line 270  FaceNodes_Tri[9]   = 2;
Line 271  FaceNodes_Tri[10]  = 5;
Line 272  FaceNodes_Tri[11]  = 6;
Line 273
Line 274  //Nodes for Face 4
Line 275  FaceNodes_Tri[12]  = 5;
Line 276  FaceNodes_Tri[13]  = 4;
Line 277  FaceNodes_Tri[14]  = 6;
Line 278
Line 279  //Nodes for Face 5
Line 280  FaceNodes_Tri[15]  = 1;
Line 281  FaceNodes_Tri[16]  = 2;
Line 282  FaceNodes_Tri[17]  = 3;

```

- [Line 251](#) and [Line 254](#)- 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.

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 [Section 2- 8.3 “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 [Line 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).

```
Line 283  INTEGER4      *FaceLeftElems_Tri,      *FaceRightElems_Tri;
Line 284
Line 285  FaceLeftElems_Tri = (INTEGER4*)malloc(NumFaces_Tri *
      sizeof(INTEGER4));
Line 286
Line 287  FaceRightElems_Tri = (INTEGER4*)malloc(NumFaces_Tri *
      sizeof(INTEGER4));
Line 288
Line 289  for(ii=0;ii<NumFaces_Tri;ii++)
Line 290  {
Line 291      FaceRightElems_Tri[ii] = 1;
Line 292      FaceLeftElems_Tri[ii] = 0;
Line 293  }
Line 294
Line 295  FaceLeftElems_Tri[0] = -1;
```

- [Line 151](#) - 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. 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.



Step 11: Specify boundary connections for Zone 2

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

```

Line 296  INTEGER4      *FaceBndryConnectionCounts_Tri,
Line 297                      *FaceBndryConnectionElems_Tri;
Line 298  INTEGER2      *FaceBndryConnectionZones_Tri;
Line 299
Line 300  FaceBndryConnectionCounts_Tri = (INTEGER4*)
        malloc(NumConnectedBndryFaces_Tri * sizeof(INTEGER4));
Line 301
Line 302  FaceBndryConnectionElems_Tri =
        (INTEGER4*)malloc(TotalNumBndryConns_Tri *
        sizeof(INTEGER4));
Line 303
Line 304  FaceBndryConnectionZones_Tri =
        (INTEGER2*)malloc(TotalNumBndryConns_Tri *
        sizeof(INTEGER2));
Line 305
Line 306  FaceBndryConnectionCounts_Tri[0] = 2;
Line 307
Line 308  FaceBndryConnectionElems_Tri[0] = 0;
Line 309  FaceBndryConnectionZones_Tri[0] = 0;
Line 310
Line 311  FaceBndryConnectionElems_Tri[1] = 1;
Line 312  FaceBndryConnectionZones_Tri[1] = 1;
Line 313
Line 314  I = TECPOLY111(FaceNodeCounts_Tri,
Line 315                      FaceNodes_Tri,
Line 316                      FaceLeftElems_Tri,
Line 317                      FaceRightElems_Tri,
Line 318                      FaceBndryConnectionCounts_Tri,
Line 319                      FaceBndryConnectionElems_Tri,
Line 320                      FaceBndryConnectionZones_Tri);
Line 321
Line 322  /* cleanup */

```



```
Line 323  free(X_Tri) ;
Line 324  free(Y_Tri) ;
Line 325  free(Z_Tri) ;
Line 326  free(P_Tri) ;
Line 327  free(FaceNodeCounts_Tri) ;
Line 328  free(FaceNodes_Tri) ;
Line 329  free(FaceLeftElems_Tri) ;
Line 330  free(FaceRightElems_Tri) ;
Line 331  free(FaceBndryConnectionCounts_Tri) ;
Line 332  free(FaceBndryConnectionElems_Tri) ;
Line 333  free(FaceBndryConnectionZones_Tri) ;
```

- [Line 308](#) and [Line 309](#) - 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's 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.
- [Line 311](#) and [Line 312](#) - Indicates that Face 1 is connected to Element 1 in Zone 1.

Step 12: Close the file

Call TECEND to close the file.

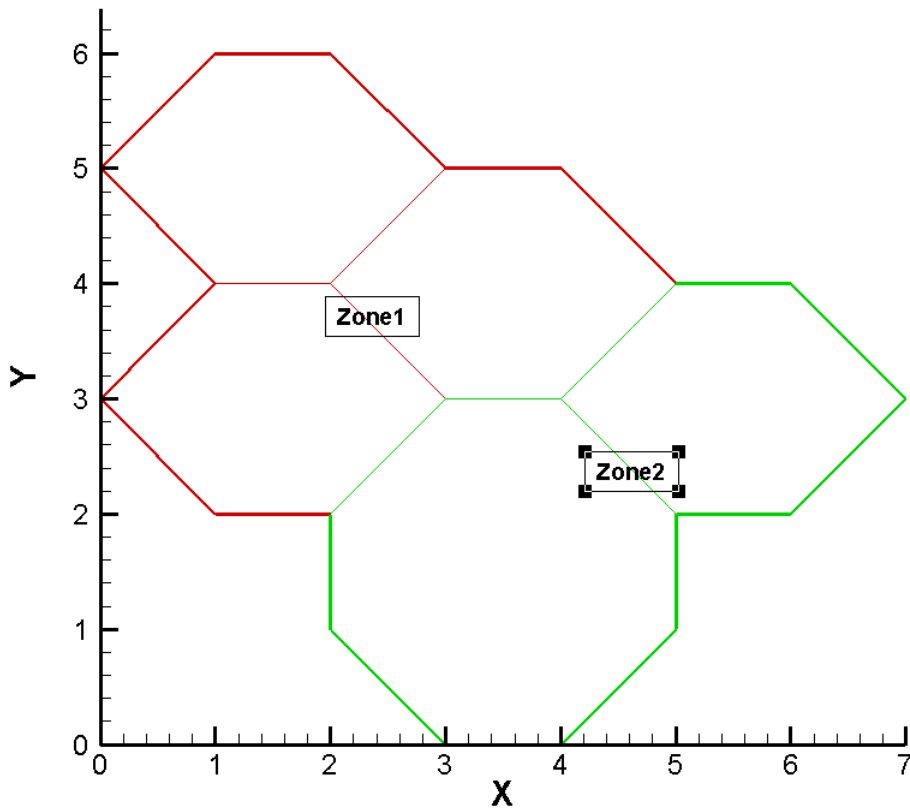
```
Line 334  I = TECEND111 () ;
```

2- 9.4 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).



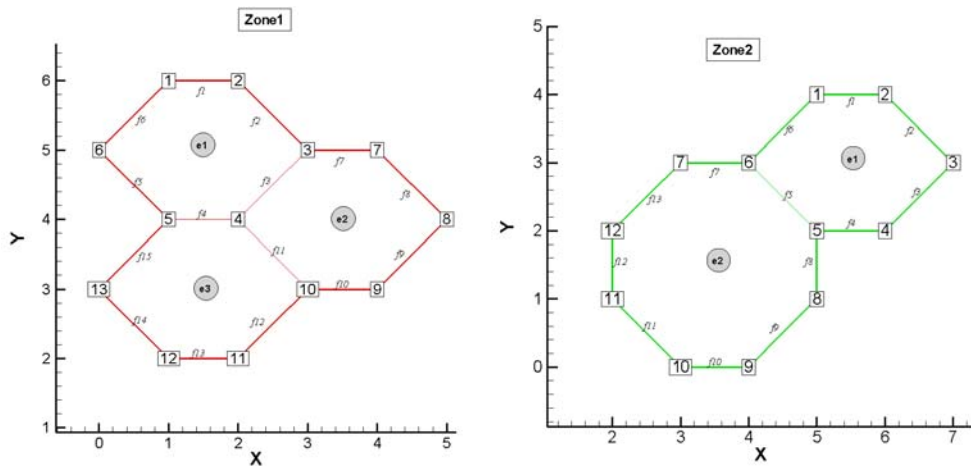
The code in this example produces the following plot:



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` and `malloc.h`¹. The source files for this example are included in your Tecplot 360 installation under the `\util\tecio\polyhedral\MultiPoly2D` subdirectory.

For complete details on the parameters used and the function syntax for each TecIO function, refer to [Section 2 - 7 “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 [Section 2 - 4 “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`.

```
Line 1  INTEGER4 Debug, I, VIsDouble, FileType;
Line 2
```

1. You may notice that `malloc` is used throughout the example. This is done to clearly indicate the dimensions required for each array. It is not required in practice.



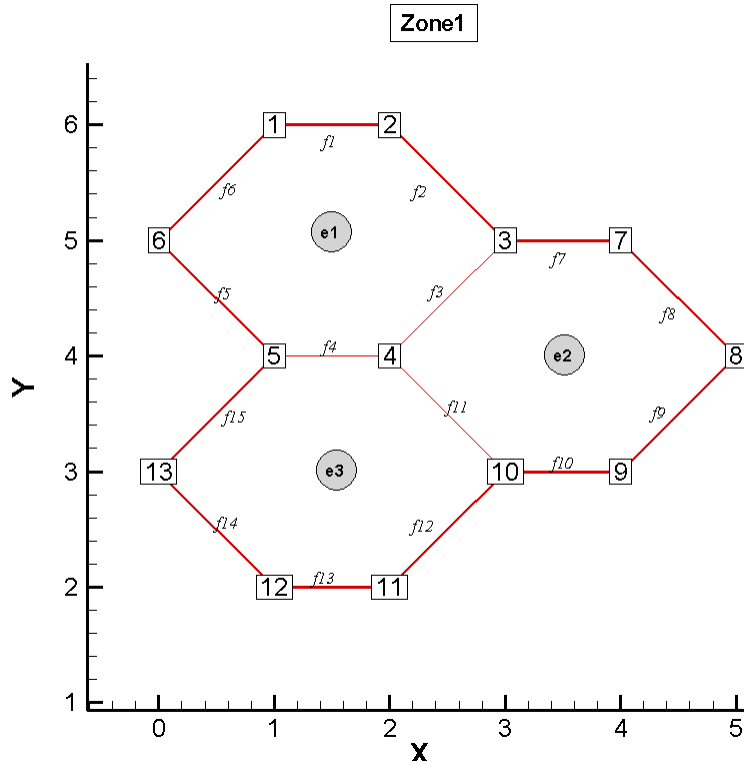
```
Line 3   Debug      = 1;  
Line 4   VisDouble = 0;  
Line 5   FileType  = 0;  
Line 6  
Line 7   I = TECINI111("Example: Multiple polygonal zones",  
Line 8                               "X Y P",  
Line 9                               "MultiPoly2D.plt",  
Line 10                              ".",  
Line 11                              &FileType,  
Line 12                              &Debug,  
Line 13                              &VisDouble);
```

- [Line 7](#) - Specifies the name of the entire data set. When the file is loaded into Tecplot, the value is available via the **Data Set Information** dialog.
- [Line 8](#) - 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 Variable 1). When referring to variables in other TecIO functions, you will refer to the variable by its number.



Step 2: Create Zone 1 (3 Hexagons)

The first step toward creating Zone 1 is to call TECZNE. TECZNE 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).



```

Line 14  INTEGER4  ZoneType, NumPts_Z1, NumElems_Z1, NumFaces_Z1,
Line 15                                     ICellMax, JCellMax, KCellMax, StrandID,
Line 16                                     ParentZone, IsBlock, NumFaceConnections,
Line 17                                     FaceNeighborMode, ValueLocation[3], SharConn,
Line 18                                     TotalNumFaceNodes_Z1, TotalNumBndryFaces_Z1,
Line 19                                     TotalNumBndryConns_Z1;
Line 20  double   SolutionTime;
Line 21
Line 22  //TECZNE Parameters

```




```
Line 23   ZoneType           = 6;
Line 24   NumPts_Z1         = 13;
Line 25   NumElems_Z1      = 3;
Line 26   NumFaces_Z1      = 15;
Line 27   ICellMax = JCellMax = KCellMax = 0;
Line 28   SolutionTime     = 0.0;
Line 29   StrandID        = 0;
Line 30   ParentZone      = 0;
Line 31   IsBlock         = 1;
Line 32   NumFaceConnections = 0;
Line 33   FaceNeighborMode = 1;
Line 34   SharConn        = 0;
Line 35   ValueLocation[0] = 1;
Line 36   ValueLocation[1] = 1;
Line 37   ValueLocation[2] = 0;
Line 38
Line 39   TotalNumFaceNodes_Z1 = 2 * NumFaces_Z1;
Line 40   TotalNumBndryFaces_Z1 = 3;
Line 41   TotalNumBndryConns_Z1 = 3;
Line 42
Line 43   I = TECZNE111("Zone 1: 3 Hexagons",
Line 44                       &ZoneType,
Line 45                       &NumPts_Z1,
Line 46                       &NumElems_Z1,
Line 47                       &NumFaces_Z1,
Line 48                       &ICellMax,
Line 49                       &JCellMax,
Line 50                       &KCellMax,
Line 51                       &SolutionTime,
Line 52                       &StrandID,
Line 53                       &ParentZone,
Line 54                       &IsBlock,
Line 55                       &NumFaceConnections,
```



<i>Line 56</i>	<code>&FaceNeighborMode,</code>
<i>Line 57</i>	<code>&TotalNumFaceNodes_Z1,</code>
<i>Line 58</i>	<code>&TotalNumBndryFaces_Z1,</code>
<i>Line 59</i>	<code>&TotalNumBndryConns_Z1,</code>
<i>Line 60</i>	<code>NULL,</code>
<i>Line 61</i>	<code>ValueLocation,</code>
<i>Line 62</i>	<code>NULL,</code>
<i>Line 63</i>	<code>&SharConn);</code>

- [Line 23](#) - Set the zone type to polygonal.
- [Line 24](#) - Specify that the zone contains 13 nodes or points.
- [Line 25](#) - Specify that the zone contains 3 elements.
- [Line 26](#) - Specify that the zone contains 15 faces.
- [Line 39](#) - 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.
- [Line 40](#) - 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”.
- [Line 41](#) - 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).

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 NumElems (if the variables are cell-centered). The variable location is specified by the VarLocation parameter in TECZNE. 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) cor-



responds 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 [Line 2](#) for the X and Y coordinate values for Zone 1.

```
Line 64    double *X_Z1, *Y_Z1;
Line 65    X_Z1 = (double*)malloc(NumPts_Z1 * sizeof(double));
Line 66    Y_Z1 = (double*)malloc(NumPts_Z1 * sizeof(double));
Line 67
Line 68    X_Z1[0] = 1;
Line 69    Y_Z1[0] = 6;
Line 70
Line 71    X_Z1[1] = 2;
Line 72    Y_Z1[1] = 6;
Line 73
Line 74    X_Z1[2] = 3;
Line 75    Y_Z1[2] = 5;
Line 76
Line 77    X_Z1[3] = 2;
Line 78    Y_Z1[3] = 4;
Line 79
Line 80    X_Z1[4] = 1;
Line 81    Y_Z1[4] = 4;
Line 82
Line 83    X_Z1[5] = 0;
Line 84    Y_Z1[5] = 5;
Line 85
Line 86    X_Z1[6] = 4;
Line 87    Y_Z1[6] = 5;
Line 88
Line 89    X_Z1[7] = 5;
Line 90    Y_Z1[7] = 4;
Line 91
```



```
Line 92   X_Z1[8] = 4;
Line 93   Y_Z1[8] = 3;
Line 94
Line 95   X_Z1[9] = 3;
Line 96   Y_Z1[9] = 3;
Line 97
Line 98   X_Z1[10] = 2;
Line 99   Y_Z1[10] = 2;
Line 100
Line 101  X_Z1[11] = 1;
Line 102  Y_Z1[11] = 2;
Line 103
Line 104  X_Z1[12] = 0;
Line 105  Y_Z1[12] = 3;
Line 106
Line 107  double *P_Z1;
Line 108  P_Z1 = (double*)malloc(NumElems_Z1 * sizeof(double));
Line 109
Line 110  P_Z1[0] = 2;
Line 111  P_Z1[1] = 4;
Line 112  P_Z1[2] = 5;
Line 113
Line 114  INTEGER4  IsDouble = 1;
Line 115
Line 116  I = TECDAT111(&NumPts_Z1, &X_Z1[0], &IsDouble);
Line 117  I = TECDAT111(&NumPts_Z1, &Y_Z1[0], &IsDouble);
Line 118  I = TECDAT111(&NumElems_Z1, &P_Z1[0], &IsDouble);
Line 119  free(X_Z1);
Line 120  free(Y_Z1);
Line 121  free(P_Z1);
```



Step 4: Specify the face map data for Zone 1

Use the picture in [Line 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.

```
Line 122  INTEGER4 *FaceNodes_Z1;
Line 123
Line 124  FaceNodes_Z1 = (INTEGER4*)malloc(TotalNumFaceNodes_Z1 *
      sizeof(INTEGER4));
Line 125
Line 126  //Face Nodes for Element 1
Line 127  FaceNodes_Z1[0] = 1;
Line 128  FaceNodes_Z1[1] = 2;
Line 129
Line 130  FaceNodes_Z1[2] = 2;
Line 131  FaceNodes_Z1[3] = 3;
Line 132
Line 133  FaceNodes_Z1[4] = 3;
Line 134  FaceNodes_Z1[5] = 4;
Line 135
Line 136  FaceNodes_Z1[6] = 4;
Line 137  FaceNodes_Z1[7] = 5;
Line 138
Line 139  FaceNodes_Z1[8] = 5;
Line 140  FaceNodes_Z1[9] = 6;
Line 141
Line 142  FaceNodes_Z1[10] = 6;
Line 143  FaceNodes_Z1[11] = 1;
Line 144
Line 145  //Face Nodes for Element 2
Line 146  FaceNodes_Z1[12] = 3;
Line 147  FaceNodes_Z1[13] = 7;
Line 148
Line 149  FaceNodes_Z1[14] = 7;
Line 150  FaceNodes_Z1[15] = 8;
```



```
Line 151
Line 152  FaceNodes_Z1[16] = 8;
Line 153  FaceNodes_Z1[17] = 9;
Line 154
Line 155  FaceNodes_Z1[18] = 9;
Line 156  FaceNodes_Z1[19] = 10;
Line 157
Line 158  FaceNodes_Z1[20] = 10;
Line 159  FaceNodes_Z1[21] = 4;
Line 160
Line 161  //Face Nodes for Element 3
Line 162  FaceNodes_Z1[22] = 10;
Line 163  FaceNodes_Z1[23] = 11;
Line 164
Line 165  FaceNodes_Z1[24] = 11;
Line 166  FaceNodes_Z1[25] = 12;
Line 167
Line 168  FaceNodes_Z1[26] = 12;
Line 169  FaceNodes_Z1[27] = 13;
Line 170
Line 171  FaceNodes_Z1[28] = 13;
Line 172  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 [Section 2- 8.3 “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 [Line 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).

```

Line 173  INTEGER4 *FaceLeftElems_Z1,   *FaceRightElems_Z1;
Line 174
Line 175  FaceLeftElems_Z1 = (INTEGER4*)malloc (NumFaces_Z1 *
        sizeof(INTEGER4)) ;
Line 176
Line 177  FaceRightElems_Z1= (INTEGER4*)malloc (NumFaces_Z1 *
        sizeof(INTEGER4)) ;
Line 178
Line 179      //Left Face Elems for Element 1
Line 180  FaceLeftElems_Z1[0] = 0;
Line 181  FaceLeftElems_Z1[1] = 0;
Line 182  FaceLeftElems_Z1[2] = 2;
Line 183  FaceLeftElems_Z1[3] = 3;
Line 184  FaceLeftElems_Z1[4] = 0;
Line 185  FaceLeftElems_Z1[5] = 0;
Line 186
Line 187      //Left Face Elems for Element 2
Line 188  FaceLeftElems_Z1[6] = 0;
Line 189  FaceLeftElems_Z1[7] = 0;
Line 190  FaceLeftElems_Z1[8] = -1;
Line 191  FaceLeftElems_Z1[9] = -2;
Line 192  FaceLeftElems_Z1[10] = 2;
Line 193
Line 194      //Left Face Elems for Element 3
Line 195  FaceLeftElems_Z1[11] = -3;
Line 196  FaceLeftElems_Z1[12] = 0;
Line 197  FaceLeftElems_Z1[13] = 0;

```



```

Line 198   FaceLeftElems_Z1[14] = 0;
Line 199
Line 200   INTEGER4   ii;
Line 201   //Set Right Face Elems
Line 202   for(ii=0;ii<6;ii++)
Line 203       FaceRightElems_Z1[ii] = 1;
Line 204
Line 205   for(ii=6;ii<10;ii++)
Line 206       FaceRightElems_Z1[ii] = 2;
Line 207
Line 208   for(ii=10;ii<=14;ii++)
Line 209       FaceRightElems_Z1[ii] = 3;

```

- [Line 190](#), [Line 191](#), [Line 195](#) - A negative value indicates that the neighboring element is in another zone. The number is a pointer into the FaceBndryConnectionElems and FaceBndryConnectionZones arrays. In this example, [Line 195](#) indicates that the left neighboring element for Face 12¹ is element two in zone two. Refer to the following step for details.

Step 6: Specify the boundary connections for Zone 1

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

```

Line 210   //Dimensioned by the TotalNumBndryFaces
Line 211   INTEGER4 FaceBndryConnectionCounts_Z1[3] = {1,1,1};
Line 212   INTEGER4 FaceBndryConnectionElems_Z1[3] = {1,2,2};
Line 213   INTEGER2 FaceBndryConnectionZones_Z1[3] = {2,2,2};
Line 214
Line 215   I = TECPOLY111(NULL,
Line 216               &FaceNodes_Z1[0],
Line 217               &FaceLeftElems_Z1[0],
Line 218               &FaceRightElems_Z1[0],
Line 219               &FaceBndryConnectionCounts_Z1[0],
Line 220               &FaceBndryConnectionElems_Z1[0],

```

1. In C, the index values are zero-based. However, Tecplot uses a 1-based numbering scheme for nodes, faces, elements and zones.




```
Line 221          &FaceBndryConnectionZones_Z1[0] );  
Line 222  
Line 223    free (FaceNodes_Z1) ;  
Line 224    free (FaceLeftElems_Z1) ;  
Line 225    free (FaceRightElems_Z1) ;
```

- [Line 211](#) - 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].

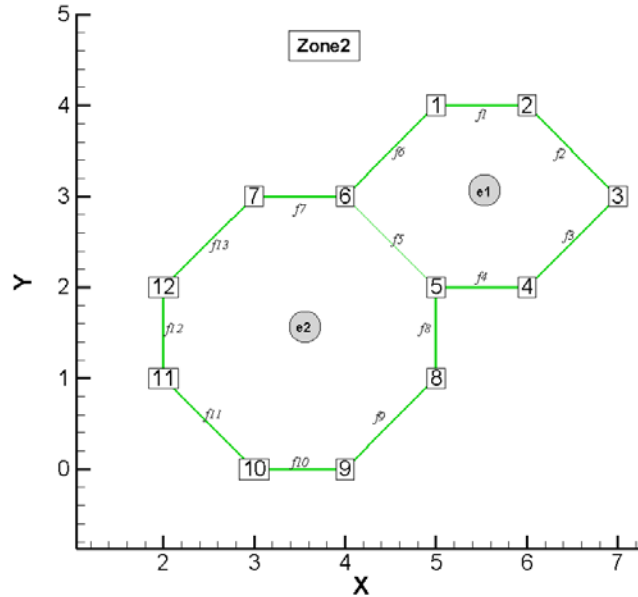
In this example, the total number of connected boundary faces (specified via TECZNE in [Line 41](#)) is equal to three. Each boundary face is connected to only one other element, so the FaceBoundaryConnectionCounts array is (1, 1, 1).

- [Line 212](#) and [Line 213](#) - 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.



Step 7: Create Zone 2

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



```

Line 226  INTEGER4  NumPts_Z2, NumElems_Z2, NumFaces_Z2,
Line 227          NumFaceConnections_Z2, TotalNumBndryFaces_Z2,
Line 228          TotalNumBndryConns_Z2, TotalNumFaceNodes_Z2;
Line 229
Line 230  NumPts_Z2          = 12;
Line 231  NumElems_Z2       = 2;
Line 232  NumFaces_Z2      = 13;
Line 233  NumFaceConnections_Z2 = 0;
Line 234  TotalNumFaceNodes_Z2 = NumFaces_Z2 * 2;
Line 235  TotalNumBndryFaces_Z2 = 3;
Line 236  TotalNumBndryConns_Z2 = 3;
Line 237
Line 238  I = TECZNE111("Zone 2: 1 Hexagon and 1 Octagon",

```



```

Line 239          &ZoneType,
Line 240          &NumPts_Z2,
Line 241          &NumElems_Z2,
Line 242          &NumFaces_Z2,
Line 243          &ICellMax,
Line 244          &JCellMax,
Line 245          &KCellMax,
Line 246          &SolutionTime,
Line 247          &StrandID,
Line 248          &ParentZone,
Line 249          &IsBlock,
Line 250          &NumFaceConnections_Z2,
Line 251          &FaceNeighborMode,
Line 252          &TotalNumFaceNodes_Z2,
Line 253          &TotalNumBndryFaces_Z2,
Line 254          &TotalNumBndryConns_Z2,
Line 255          NULL,
Line 256          ValueLocation,
Line 257          NULL,
Line 258          &SharConn) ;

```

- [Line 230](#) - Specify that the zone contains 12 nodes or points.
- [Line 231](#) - Specify that the zone contains 2 elements.
- [Line 232](#) - Specify that the zone contains 13 faces.
- [Line 235](#) - 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”.
- [Line 236](#) - 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).



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 NumElems (if the variables are cell-centered). The variable location is specified by the VarLocation parameter in TECZNE. 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 [Line 7](#) for the X and Y coordinate values for Zone 2.

```
Line 259  double   *X_Z2, *Y_Z2;
Line 260
Line 261  X_Z2 = (double*) malloc(NumPts_Z2 * sizeof(double));
Line 262  Y_Z2 = (double*) malloc(NumPts_Z2 * sizeof(double));
Line 263
Line 264  X_Z2[0] = 5;
Line 265  X_Z2[1] = 6;
Line 266  X_Z2[2] = 7;
Line 267  X_Z2[3] = 6;
Line 268  X_Z2[4] = 5;
Line 269  X_Z2[5] = 4;
Line 270  X_Z2[6] = 3;
Line 271  X_Z2[7] = 5;
Line 272  X_Z2[8] = 4;
Line 273  X_Z2[9] = 3;
Line 274  X_Z2[10] = 2;
Line 275  X_Z2[11] = 2;
Line 276
Line 277  Y_Z2[0] = 4;
Line 278  Y_Z2[1] = 4;
```



```

Line 279  Y_Z2[2] = 3;
Line 280  Y_Z2[3] = 2;
Line 281  Y_Z2[4] = 2;
Line 282  Y_Z2[5] = 3;
Line 283  Y_Z2[6] = 3;
Line 284  Y_Z2[7] = 1;
Line 285  Y_Z2[8] = 0;
Line 286  Y_Z2[9] = 0;
Line 287  Y_Z2[10] = 1;
Line 288  Y_Z2[11] = 2;
Line 289
Line 290  double  *P_Z2;
Line 291  P_Z2 = (double*) malloc(NumPts_Z2 * sizeof(double));
Line 292
Line 293  P_Z2[0] = 8;
Line 294  P_Z2[1] = 6;
Line 295
Line 296  I = TECDAT111 (&NumPts_Z2,   &X_Z2[0], &IsDouble);
Line 297  I = TECDAT111 (&NumPts_Z2,   &Y_Z2[0], &IsDouble);
Line 298  I = TECDAT111 (&NumElems_Z2, &P_Z2[0], &IsDouble);
Line 299
Line 300  free(X_Z2);
Line 301  free(Y_Z2);
Line 302  free(P_Z2);

```

Step 9: Specify the face map for Zone 2

Use the picture in [Line 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.

```

Line 303  INTEGER4 *FaceNodes_Z2;
Line 304
Line 305  FaceNodes_Z2 = (INTEGER4*) malloc(TotalNumFaceNodes_Z2 *
                                     sizeof(INTEGER4));
Line 306

```



```
Line 307 //Face Nodes for Element 1
Line 308 FaceNodes_Z2[0] = 1;
Line 309 FaceNodes_Z2[1] = 2;
Line 310
Line 311 FaceNodes_Z2[2] = 2;
Line 312 FaceNodes_Z2[3] = 3;
Line 313
Line 314 FaceNodes_Z2[4] = 3;
Line 315 FaceNodes_Z2[5] = 4;
Line 316
Line 317 FaceNodes_Z2[6] = 4;
Line 318 FaceNodes_Z2[7] = 5;
Line 319
Line 320 FaceNodes_Z2[8] = 5;
Line 321 FaceNodes_Z2[9] = 6;
Line 322
Line 323 FaceNodes_Z2[10] = 6;
Line 324 FaceNodes_Z2[11] = 1;
Line 325
Line 326
Line 327 //Face Nodes for Element 2
Line 328 FaceNodes_Z2[12] = 7;
Line 329 FaceNodes_Z2[13] = 6;
Line 330
Line 331 FaceNodes_Z2[14] = 5;
Line 332 FaceNodes_Z2[15] = 8;
Line 333
Line 334 FaceNodes_Z2[16] = 8;
Line 335 FaceNodes_Z2[17] = 9;
Line 336
Line 337 FaceNodes_Z2[18] = 9;
Line 338 FaceNodes_Z2[19] = 10;
Line 339
```



```

Line 340  FaceNodes_Z2 [20] = 10;
Line 341  FaceNodes_Z2 [21] = 11;
Line 342
Line 343  FaceNodes_Z2 [22] = 11;
Line 344  FaceNodes_Z2 [23] = 12;
Line 345
Line 346  FaceNodes_Z2 [24] = 12;
Line 347  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 [Section 2- 8.3 “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 [Line 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).

```

Line 348  INTEGER4 *FaceLeftElems_Z2,  *FaceRightElems_Z2;
Line 349
Line 350  FaceLeftElems_Z2 = (INTEGER4*) malloc (NumFaces_Z2 *
sizeOf (INTEGER4));
Line 351  FaceRightElems_Z2 = (INTEGER4*) malloc (NumFaces_Z2 *
sizeOf (INTEGER4));
Line 352
Line 353  //Left Face Elems for Element 1
Line 354  FaceLeftElems_Z2 [4] = 2;

```



```

Line 355  FaceLeftElems_Z2[5]  = -1;
Line 356  FaceLeftElems_Z2[6]  = -2;
Line 357  FaceLeftElems_Z2[12] = -3;
Line 358
Line 359  //Set Right Face Elems
Line 360  for(ii=0;ii<6;ii++)
Line 361      FaceRightElems_Z2[ii] = 1;
Line 362
Line 363  for(ii=6;ii<13;ii++)
Line 364      FaceRightElems_Z2[ii] = 2;
Line 365
Line 366  //Set Left Face Elems that are "no neighboring element"
Line 367  INTEGER4 FacesWithNoNeighboringElements_Z2[9] =
Line 368      { 1, 2, 3, 4,          //Faces in Element 1
Line 369      8, 9, 10, 11, 12}; //Faces in Element 2
Line 370
Line 371  for (ii=0;ii<9;ii++)
Line 372  FaceLeftElems_Z2[FacesWithNoNeighboringElements_Z2[ii]-1]=0;

```

- [Line 355](#), [Line 356](#), [Line 357](#) - A negative value indicates that the neighboring element is in another zone. The number is a pointer into the FaceBndryConnectionElems and FaceBndryConnectionZones arrays. In this example, [Line 357](#) indicates that the left element of Face 13 is element three in zone one. Refer to the following step for details.
- [Line 366](#) - [Line 372](#) - For brevity we have explicitly defined which faces have “no neighboring element” (i.e. are on the edge of the data set) and used a for loop to set the neighboring element value to “0”.

Step 11: Specify the Boundary Connections for Zone 2

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

```

Line 1    INTEGER4 FaceBndryConnectionCounts_Z2[3]  = {1,1,1};
Line 2    INTEGER4 FaceBndryConnectionElems_Z2[3]  = {2,3,3};
Line 3    INTEGER2 FaceBndryConnectionZones_Z2[3]  = {1,1,1};
Line 4

```




```

Line 5      I = TECPOLY111 (NULL,
Line 6                      &FaceNodes_Z2 [0] ,
Line 7                      &FaceLeftElems_Z2 [0] ,
Line 8                      &FaceRightElems_Z2 [0] ,
Line 9                      &FaceBndryConnectionCounts_Z2 [0] ,
Line 10                     &FaceBndryConnectionElems_Z2 [0] ,
Line 11                     &FaceBndryConnectionZones_Z2 [0] ) ;
Line 12
Line 13     free (FaceNodes_Z2) ;
Line 14     free (FaceLeftElems_Z2) ;
Line 15     free (FaceRightElems_Z2) ;

```

- [Line 1](#) - 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 TECZNE in [Line 235](#)) is equal to three. Each boundary face is connected to only one other element, so the FaceBoundaryConnectionCounts array is (1, 1, 1).
- [Line 2](#) and [Line 3](#) - 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.

Step 12: Close the file using TECEND

Call TECEND to close the file.

```

Line 16     I = TECEND111 () ;

```

2- 9.5 Polyhedral Example

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

```

Line 17     #include "TECIO.h"
Line 18     #include "MASTER.h"
Line 19

```



```
Line 20 #define NUMNODES 6
Line 21 #define NUMELEMENTS 1
Line 22 #define NUMFACES 8
Line 23 #define NUMFACENODES 24
Line 24
Line 25 int main()
Line 26 {
Line 27     /* Declare Variables */
Line 28     double    SolutionTime;
Line 29     INTEGER4  FileType, Debug, VIsDouble;
Line 30     INTEGER4  *VarShareArray, ShrConn, DIsDouble;
Line 31     INTEGER4  ZoneType, IMaxOrNumNodes, JMaxOrNumElems,
KMaxOrNumFaces;
Line 32     INTEGER4  ICellMax, JCellMax, KCellMax, StrandID,
ParentZone,
Line 33     INTEGER4  IsBlock;
Line 34     INTEGER4  NFConns, FNMode, *PassiveVarArray,
*ValueLocArray;
Line 35     INTEGER4  IsOk, NumFaceNodes, NumBConns, NumBItems;
Line 36
Line 37     /* Initialize arrays of nodal data */
Line 38     double    X[NUMNODES] = { 0, 1, 0, -1, 0, 0 };
Line 39     double    Y[NUMNODES] = { -1, 0, 1, 0, 0, 0 };
Line 40     double    Z[NUMNODES] = { 0, 0, 0, 0, 1, -1 };
Line 41
Line 42     /*
Line 43     * Initialize Face Map Arrays:
Line 44     * FaceNodes holds face nodes in the order face1, face2...
Line 45     * FaceLeftElems and FaceRightElems hold the one-based
element
Line 46     * numbers for the right and left elements for a given
face.
Line 47     * Ordered face1, face2 ...
Line 48     * FaceNodeCounts gives the number of nodes in each face
```



```

Line 49      */
Line 50      INTEGER4 FaceNodes [NUMFACENODES] = {1, 2, 5, 2, 3, 5, 3,
Line 51      4, 5,
Line 52      4, 1, 5, 1, 2, 6, 2,
Line 53      3, 6,
Line 54      3, 4, 6, 4, 1, 6};
Line 55      INTEGER4 FaceLeftElems [NUMFACES] = {1,1,1,1,0,0,0,0 };
Line 56      INTEGER4 FaceRightElems [NUMFACES] = { 0,0,0,0,1,1,1,1 };
Line 57      INTEGER4 FaceNodeCounts [NUMFACES] = { 3,3,3,3,3,3,3,3 };
Line 58      /* Set the variable that will be used for error tracking.*/
Line 59      IsOk = TRUE;
Line 60
Line 61      /* Call TECINI111 */
Line 62      FileType = 0; /* 0 for full file, 1 for grid file,
Line 63      * 2 for solution file*/
Line 64      Debug = 1; /* 0 for no debug output, 1 to show debug */
Line 65      VIsDouble = 1; /* 0 for single precision,
Line 66      1 for double precision */
Line 67
Line 68      IsOk = TECINI111("Test Polyhedral Data Set",
Line 69      "X Y Z", /* Variable List */
Line 70      "simplepolyhedron.plt", /* File Name */
Line 71      ".", /* Scratch Dir */
Line 72      &(FileType),
Line 73      &(Debug),
Line 74      &(VIsDouble));
Line 75
Line 76
Line 77      /* Call TECZNE111 */
Line 78      ZoneType = 7; /* 7 for FEPolyhedron */
Line 79      IMaxOrNumNodes = NUMNODES; /* Num of nodes */
Line 80      JMaxOrNumElems = NUMELEMENTS; /* Num of elements */

```



```

Line 81      KMaxOrNumFaces = NUMFACES; /* Num of Faces */
Line 82      ICellMax      = 0;          /* Not Used, set to zero */
Line 83      JCellMax      = 0;          /* Not Used, set to zero */
Line 84      KCellMax      = 0;          /* Not Used, set to zero */
Line 85      SolutionTime = 12.65; /* The solution time for the zone */
Line 86      StrandID     = 0;          /* The strandid for the zone,
Line 87                                     * zero for static zones */
Line 88      ParentZone   = 0;          /* The ParentZone for this zone,
Line 89                                     * zero for no parent */
Line 90      IsBlock      = 1;          /* One for passing the data one
Line 91                                     * variable at a time, Zero for
Line 92                                     * passing the data one point
Line 93                                     * at a time */
Line 94      NFConns      = 0;          /* Number of face neighbor
Line 95                                     * connections, not used
Line 96                                     * for FEPolyhedron zones
*/
Line 97      FNMode        = 0; /* Not used for Polyhedron zones */
Line 98      PassiveVarArray = NULL;    /* No passive variables */
Line 99      ValueLocArray  = NULL;    /* All nodal variables */
Line 100     VarShareArray  = NULL;    /* No variable sharing */
Line 101     ShrConn         = 0;          /* No connectivity sharing */
Line 102     NumFaceNodes    = NUMFACENODES; /* The num of face nodes */
Line 103     NumBConns       = 0;          /* No Boundary Connections */
Line 104     NumBItems        = 0;          /* No Boundary Items */
Line 105
Line 106     IsOk = TECZNE111("Polyhedral Zone (Octahedron)",
Line 107                       &(ZoneType) ,
Line 108                       &(IMaxOrNumNodes) ,
Line 109                       &(JMaxOrNumElems) ,
Line 110                       &(KMaxOrNumFaces) ,
Line 111                       &ICellMax ,
Line 112                       &JCellMax ,

```



```

Line 113             &KCellMax,
Line 114             &SolutionTime,
Line 115             &StrandID,
Line 116             &ParentZone,
Line 117             &IsBlock,
Line 118             &NFConns,
Line 119             &FNMode,
Line 120             &NumFaceNodes,
Line 121             &NumBConns,
Line 122             &NumBItems,
Line 123             PassiveVarArray,
Line 124             ValueLocArray,
Line 125             VarShareArray,
Line 126             &ShrConn);
Line 127
Line 128
Line 129             /* Write the data (using TECDAT111) */
Line 130             DIIsDouble = 1; /* double precision */
Line 131             IsOk = TECDAT111(&IMaxOrNumNodes, &(X[0]),
Line 132             &(DIIsDouble));
Line 133             IsOk = TECDAT111(&IMaxOrNumNodes, &(Y[0]),
Line 134             &(DIIsDouble));
Line 135             IsOk = TECDAT111(&IMaxOrNumNodes, &(Z[0]),
Line 136             &(DIIsDouble));
Line 137
Line 138             /* Write the face map (created above) using TECPOLY111. */
Line 139             IsOk = TECPOLY111(&(FaceNodeCounts[0]),
Line 140             &(FaceNodes[0]),
Line 141             &(FaceLeftElems[0]),
Line 142             &(FaceRightElems[0]),
Line 143             NULL,
Line 144             NULL,
Line 145             NULL);
Line 146             IsOk = TECEND111();

```



```
Line 144     return 0;
Line 145     }
Line 146
```

2- 9.6 IJ-ordered zone

The following example illustrates how to create a simple IJ-ordered zone. [TECZNE111](#) is called first to initialize the zone.

```
Line 147 #include <stdio.h>
Line 148 #include <string.h>
Line 149 #include "TECIO.h"
Line 150 int main ()
Line 151 {
Line 152     float      X[2][2], Y[2][2], P[2][2];
Line 153     double      SolTime;
Line 154     INTEGER4    Debug, I, III, DIsDouble, VIsDouble, IMax, JMax;
Line 155     INTEGER4    KMax, ZoneType, StrandID, ParentZn, IsBlock;
Line 156     INTEGER4    ICellMax, JCellMax, KCellMax;
Line 157     INTEGER4    NFConns, FNMode, ShrConn, FileType;
Line 158     INTEGER4    TotalNumFaceNodes, TotalNumBndryFaces,
Line 159                TotalNumBoundaryConnections;
Line 160
Line 161     Debug      = 1;
Line 162     VIsDouble  = 0;
Line 163     DIsDouble  = 0;
Line 164     IMax       = 2;
Line 165     JMax       = 2;
Line 166     KMax       = 1;
Line 167     ZoneType   = 0;      /* Ordered Zone */
Line 168     SolTime    = 360.0;
Line 169     StrandID   = 0;      /* Static Zone */
Line 170     ParentZn   = 0;      /* No Parent */
Line 171     IsBlock    = 1;      /* Block */
Line 172     ICellMax   = 0;
```



```
Line 173  JCellMax  = 0;
Line 174  KCellMax  = 0;
Line 175  NFCOnns   = 0;
Line 176  FNMode    = 0;
Line 177  TotalNumFaceNodes      = 1;
Line 178  TotalNumBndryFaces     = 1;
Line 179  TotalNumBoundaryConnections = 1;
Line 180  ShrConn   = 0;
Line 181  FileType  = 0;
Line 182
Line 183 /* Variable Values */
Line 184
Line 185  X[0][0] = .125;
Line 186  X[1][0] = .625;
Line 187  X[0][1] = .125;
Line 188  X[1][1] = .625;
Line 189
Line 190  Y[0][0] = .5;
Line 191  Y[1][0] = .5;
Line 192  Y[0][1] = .875;
Line 193  Y[1][1] = .875;
Line 194
Line 195  P[0][0] = 5;
Line 196  P[1][0] = 7.5;
Line 197  P[0][1] = 10;
Line 198  P[1][1] = 7.5;
Line 199 /*
Line 200 * Open the file and write the tecplot datafile header
Line 201 * information
Line 202 */
Line 203  I = TECINI111("SIMPLE DATASET",
Line 204                "X Y P",
Line 205                "SimpleOrderedZone.plt",
```



```
Line 206         ". ",
Line 207         &FileType,
Line 208         &Debug,
Line 209         &VIsDouble);
Line 210   I = TECZNE111("Ordered Zone",
Line 211         &ZoneType,
Line 212         &IMax,
Line 213         &JMax,
Line 214         &KMax,
Line 215         &ICellMax,
Line 216         &JCellMax,
Line 217         &KCellMax,
Line 218         &SolTime,
Line 219         &StrandID,
Line 220         &ParentZn,
Line 221         &IsBlock,
Line 222         &NFConns,
Line 223         &FNMode,
Line 224         &TotalNumFaceNodes,
Line 225         &TotalNumBndryFaces,
Line 226         &TotalNumBoundaryConnections,
Line 227         NULL,
Line 228         NULL,
Line 229         NULL,
Line 230         &ShrConn);
Line 231   III = IMax * JMax * KMax;
Line 232   I   = TECDAT111 (&III, &X[0][0], &DIsDouble);
Line 233   I   = TECDAT111 (&III, &Y[0][0], &DIsDouble);
Line 234   I   = TECDAT111 (&III, &P[0][0], &DIsDouble);
Line 235
Line 236   I = TECEND111();
Line 237
Line 238   return 0;
```




```
Line 239 }
```

```
Line 240
```

2- 9.7 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.

```
Line 1  int main ()
Line 2  {
Line 3  /* initialize file 1 */
Line 4  INTEGER4 Debug, I, VIsDouble, FileType;
Line 5  float X2[2][2], Y2[2][2], P2[2][2];
Line 6  float X3[2][2], Y3[2][2], P3[2][2];
Line 7  double SolTime;
Line 8  INTEGER4 DIsDouble, III, IMax, JMax, KMax, ZoneType;
Line 9  INTEGER4 StrandID, ParentZn, IsBlock;
Line 10  INTEGER4 ICellMax, JCellMax, KCellMax, NFConns, FNMode;
Line 11  INTEGER4 ShrConn, TotalNumFaceNodes, TotalNumBndryFaces,
Line 12  TotalNumBoundaryConnections;
Line 13  char DeformationValue[128];
Line 14  strcpy(DeformationValue,"0.98");
Line 15
Line 16  ICellMax   = 0;
Line 17  JCellMax   = 0;
Line 18  KCellMax   = 0;
Line 19  DIsDouble  = 0;
Line 20  SolTime    = 360.0;
Line 21  StrandID   = 0;    /* Static Zone */
Line 22  ParentZn   = 0;
Line 23  IsBlock    = 1;    /* Block */
Line 24  NFConns    = 0;
Line 25  FNMode     = 0;
Line 26  TotalNumFaceNodes = 1;
```



```
Line 27  TotalNumBndryFaces          = 1;  
Line 28  TotalNumBoundaryConnections = 1;  
Line 29  ShrConn      = 0;  
Line 30  /* Ordered Zone Parameters */  
Line 31  IMax = 2;  
Line 32  JMax = 2;  
Line 33  KMax = 1;  
Line 34  
Line 35  X2[0][0] = .125;  
Line 36  X2[1][0] = .625;  
Line 37  X2[0][1] = .125;  
Line 38  X2[1][1] = .625;  
Line 39  
Line 40  Y2[0][0] = .5;  
Line 41  Y2[1][0] = .5;  
Line 42  Y2[0][1] = .875;  
Line 43  Y2[1][1] = .875;  
Line 44  
Line 45  P2[0][0] = 5;  
Line 46  P2[1][0] = 7.5;  
Line 47  P2[0][1] = 10;  
Line 48  P2[1][1] = 7.5;  
Line 49  
Line 50  X3[0][0] = .375;  
Line 51  X3[1][0] = .875;  
Line 52  X3[0][1] = .375;  
Line 53  X3[1][1] = .875;  
Line 54  
Line 55  Y3[0][0] = .125;  
Line 56  Y3[1][0] = .125;  
Line 57  Y3[0][1] = .5;  
Line 58  Y3[1][1] = .5;  
Line 59
```



```
Line 60   P3[0][0] = 5;
Line 61   P3[1][0] = 7.5;
Line 62   P3[0][1] = 10;
Line 63   P3[1][1] = 7.5;
Line 64
Line 65
Line 66   Debug      = 1;
Line 67   VIsDouble = 0;
Line 68   FileType  = 0;
Line 69
Line 70
Line 71  /*
Line 72  * Open the file and write the tecplot datafile
Line 73  * header information
Line 74  */
Line 75   I = TECINI111("SIMPLE DATASET",
Line 76                   "X Y P",
Line 77                   "file1.plt",
Line 78                   ".",
Line 79                   &FileType,
Line 80                   &Debug,
Line 81                   &VIsDouble);
Line 82
Line 83
Line 84   /* Ordered Zone */
Line 85   ZoneType = 0;
Line 86   I = TECZNE111("Ordered Zone",
Line 87                   &ZoneType,
Line 88                   &IMax,
Line 89                   &JMax,
Line 90                   &KMax,
Line 91                   &ICellMax,
Line 92                   &JCellMax,
```



```
Line 93          &KCellMax,
Line 94          &SolTime,
Line 95          &StrandID,
Line 96          &ParentZn,
Line 97          &IsBlock,
Line 98          &NFConns,
Line 99          &FNMode,
Line 100         &TotalNumFaceNodes,
Line 101         &TotalNumBndryFaces,
Line 102         &TotalNumBoundaryConnections,
Line 103         NULL,
Line 104         NULL,
Line 105         NULL,
Line 106         &ShrConn) ;
Line 107  III = IMax * JMax * KMax;
Line 108  I   = TECDAT111 (&III, &X2[0][0], &DIsDouble) ;
Line 109  I   = TECDAT111 (&III, &Y2[0][0], &DIsDouble) ;
Line 110  I   = TECDAT111 (&III, &P2[0][0], &DIsDouble) ;
Line 111
Line 112  I = TECINI111 ("Auxiliary Data",
Line 113          "X Y P",
Line 114          "file2.plt",
Line 115          ".",
Line 116          &FileType,
Line 117          &Debug,
Line 118          &VIsDouble) ;
Line 119  III = 2 ;
Line 120  I = TECFIL111 (&III) ;
Line 121
Line 122  I = TECAUXSTR111 ("DeformationValue",
Line 123          DeformationValue) ;
Line 124
Line 125  I = TECZNE111 ("Ordered Zone2",
```



```

Line 126          &ZoneType,
Line 127          &IMax,
Line 128          &JMax,
Line 129          &KMax,
Line 130          &ICellMax,
Line 131          &JCellMax,
Line 132          &KCellMax,
Line 133          &SolTime,
Line 134          &StrandID,
Line 135          &ParentZn,
Line 136          &IsBlock,
Line 137          &NFConns,
Line 138          &FNMode,
Line 139          &TotalNumFaceNodes,
Line 140          &TotalNumBndryFaces,
Line 141          &TotalNumBoundaryConnections,
Line 142          NULL,
Line 143          NULL,
Line 144          NULL,
Line 145          &ShrConn);
Line 146  III = IMax * JMax * KMax;
Line 147  I   = TECDAT111 (&III, &X3[0][0], &DIsDouble);
Line 148  I   = TECDAT111 (&III, &Y3[0][0], &DIsDouble);
Line 149  I   = TECDAT111 (&III, &P3[0][0], &DIsDouble);
Line 150      III = 2;
Line 151  I = TECFIL111 (&III);
Line 152  I = TECEND();
Line 153
Line 154
Line 155  III = 1;
Line 156  I = TECFIL111 (&III);
Line 157  I = TECAUXSTR111 ("DeformationValue",
Line 158          DeformationValue);

```



```
Line 159  
Line 160   I = TECEND111 ();  
Line 161   return 0;
```

2- 9.8 Text Example

The following example creates a data file with a single text box reading “Sample Text”.

```
Line 1   int main ()  
Line 2   {  
Line 3       double   XPos, YPos, ZPos, FontHeight, BoxMargin,  
Line 4               BoxLineThickness, Angle, LineSpacing;  
Line 5       INTEGER4 Debug, I, VIsDouble, FileType, PosCoordMode,  
Line 6               AttachToZone, Zone, Font, FontHeightUnits, BoxType,  
Line 7               BoxColor, BoxFillColor, Anchor, TextColor, Scope,  
Line 8               Clipping;  
Line 9       char     Text[60], MFC[24];  
Line 10  
Line 11      /* set file parameters */  
Line 12      Debug     = 1;  
Line 13      VIsDouble = 0;  
Line 14      FileType  = 0;  
Line 15  
Line 16      /* set TECTXT parameters */  
Line 17      XPos       = 0.0;  
Line 18      YPos       = 1.0;  
Line 19      ZPos       = 2.0;  
Line 20      PosCoordMode = 0;  
Line 21      AttachToZone = 0;  
Line 22      Zone       = 2;  
Line 23      Font       = 1;  
Line 24      FontHeightUnits = 2;  
Line 25      FontHeight  = 18;  
Line 26      BoxType    = 1;
```



```
Line 27   BoxMargin           = .5;
Line 28   BoxLineThickness = .1;
Line 29   BoxColor         = 0;
Line 30   BoxFillColor    = 1;
Line 31   Angle           = 30;
Line 32   Anchor          = 1;
Line 33   LineSpacing     = 1.5;
Line 34   TextColor       = 7;
Line 35   Scope          = 1;
Line 36   Clipping        = 1;
Line 37   strcpy(Text,"Sample Text");
Line 38   strcpy(MFC,"My Macro");
Line 39   /*
Line 40   * Open the file and write the tecplot datafile
Line 41   * header information
Line 42   */
Line 43   I = TECINI111("SIMPLE DATASET",
Line 44                   "X Y P",
Line 45                   "textgeom.plt",
Line 46                   ".",
Line 47                   &FileType,
Line 48                   &Debug,
Line 49                   &VIsDouble);
Line 50   I = TECTXT111(&XPos,
Line 51                   &YPos,
Line 52                   &ZPos,
Line 53                   &PosCoordMode,
Line 54                   &AttachToZone,
Line 55                   &Zone,
Line 56                   &Font,
Line 57                   &FontHeightUnits,
Line 58                   &FontHeight,
Line 59                   &BoxType,
```



```
Line 60         &BoxMargin,  
Line 61         &BoxLineThickness,  
Line 62         &BoxColor,  
Line 63         &BoxFillColor,  
Line 64         &Angle,  
Line 65         &Anchor,  
Line 66         &LineSpacing,  
Line 67         &TextColor,  
Line 68         &Scope,  
Line 69         &Clipping,  
Line 70         &Text[0],  
Line 71         &MFC[0]);  
Line 72     I = TECEND111();  
Line 73     return 0;  
Line 74 }  
Line 75
```

2- 9.9 Geometry Example

The following example creates a data file with a single square geometry and no zones.

```
Line 76 int main ()  
Line 77 {  
Line 78     INTEGER4 Debug, I, VisDouble, FileType;  
Line 79     double   XPos, YPos, ZPos, PatternLength, LineThick;  
Line 80     double   ArrowSize, ArrowAngle;  
Line 81     INTEGER4 I, PosCoordMode, AttachToZone, Color, FillColor;  
Line 82     INTEGER4 IsFilled, GeomType, LinePattern, Scope, Clipping;  
Line 83     INTEGER4 Zone, NumPts, ArrowStyle, ArrowAttach;  
Line 84     INTEGER4 NumSegments, NumSegPts;  
Line 85     float   XGeomData, YGeomData, ZGeomData;  
Line 86     char    MFC[128];  
Line 87  
Line 88     XPos = 1.0;
```




```
Line 89   YPos = 2.0;
Line 90   ZPos = 3.0;
Line 91
Line 92   PosCoordMode = 0;
Line 93   AttachToZone = 0;
Line 94   Zone          = 1;
Line 95   Color         = 0;
Line 96   FillColor    = 6;
Line 97   IsFilled     = 1;
Line 98   GeomType     = 2;
Line 99   LinePattern  = 5;
Line 100  PatternLength = .1;
Line 101  LineThick    = .2;
Line 102  NumPts       = 100;
Line 103
Line 104  ArrowStyle    = 1;
Line 105  ArrowAttach  = 0;
Line 106  ArrowSize    = 1;
Line 107  ArrowAngle   = 30;
Line 108
Line 109  Scope        = 1;
Line 110  Clipping     = 1;
Line 111
Line 112  NumSegments  = 15;
Line 113  NumSegPts    = 25;
Line 114
Line 115  XGeomData     = 4;
Line 116  YGeomData     = 6;
Line 117  ZGeomData     = 8;
Line 118
Line 119  strcpy(MFC, "SQUARE");
Line 120  Debug         = 1;
Line 121  VIsDouble    = 0;
```



```
Line 122  FileType = 0;
Line 123
Line 124
Line 125 /*
Line 126 * Open the file and write the tecplot datafile header
      information */
Line 127  I = TECINI111("SQUARE GEOMETRY",
Line 128                "X Y P",
Line 129                "square.plt",
Line 130                ".",
Line 131                &FileType,
Line 132                &Debug,
Line 133                &VisDouble);
Line 134
Line 135  I = TECGEO111(&XPos,
Line 136                &YPos,
Line 137                &ZPos,
Line 138                &PosCoordMode,
Line 139                &AttachToZone,
Line 140                &Zone,
Line 141                &Color,
Line 142                &FillColor,
Line 143                &IsFilled,
Line 144                &GeomType,
Line 145                &LinePattern,
Line 146                &PatternLength,
Line 147                &LineThick,
Line 148                &NumPts,
Line 149                &ArrowStyle,
Line 150                &ArrowAttach,
Line 151                &ArrowSize,
Line 152                &ArrowAngle,
Line 153                &Scope,
```



```
Line 154          &Clipping,  
Line 155          &NumSegments,  
Line 156          &NumSegPts,  
Line 157          &XGeomData,  
Line 158          &YGeomData,  
Line 159          &ZGeomData,  
Line 160          MFC);  
Line 161  I = TECEND111();  
Line 162  
Line 163  return 0;  
Line 164 }  
Line 165
```





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 [Chapter 2 "Binary Data"](#) for information on creating files in Tecplot's binary format.

3 - 1 Preplot

Tecplot 360 or Preplot converts ASCII data files to binary. See [Section 4 - 15 "Tecplot-Format Loader"](#) in the [User's Manual](#) for converting with Tecplot 360, or [Section 3 - 6 "ASCII Data File Conversion to Binary"](#) for converting with Preplot. A description of the binary format is included in [Appendix A "Binary Data File Format"](#). Finally, if your data is generated in FORTRAN or C, you may be able to generate binary data files directly using the utilities described in [Chapter 2 "Binary Data"](#).

Alternatively, you may write your own Tecplot data loader using Tecplot 360's Add-on Developer's Kit (ADK). Refer to [Chapter 22 "Creating a Data Loader"](#) in the [ADK User's Manual](#) for details.

3 - 2 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 up to 32,700 zone records, 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"
```

3 - 3 ASCII File Structure

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 optional zone records containing the plot data. Zone records may contain ordered or finite-element data. Refer to [Chapter 3 “Data Structure”](#) in 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**, **DATASET**, **TAUXDATA**, or **VARAUXDATA**.

Primary Components of ASCII Data Files

[File Header](#)

[Zone Record](#)

[Text Record](#)

[Geometry Record](#)

[Custom Labels Record](#)

[Data Set Auxiliary Data Record](#)

[Variable Auxiliary Data Record](#)

3- 3.1 File Header

The File Header is an optional component of an ASCII data file. It may contain a [TITLE](#), [FILE-TYPE](#) 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	= "VARIABLE1", "VARIABLE2", "VARIABLE3", ..., "VARIABLEN"	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
```



3- 3.2 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” on page 138 for details.
Data	The data section follows the zone header. The arrangement of the data is dependent upon the values of DATA-PACKING and VARLOCATION (specified in the Zone Header). Refer to “Data” on page 142 for details.
Zone Footer	<p>The contents required for the Zone Footer depend upon the ZONETYPE (specified in the Zone Header).</p> <p>For ordered zones, the Zone Footer contains the Face Neighbor Connections List information (if any).</p> <p>For cell-based finite-element zones (FETRIANGLE, FEQUADILATERAL, FETETRAHEDRAL and FEBRICK), the Zone Footer contains Connectivity information, followed by Face Neighbor Connections List.</p> <p>For face-based finite-element zones (FEPOLYHEDRAL, FEPOLYGON), the Zone Footer contains Facemap Data, followed by Boundary Map Data.</p> <p>Refer to “Zone Footer” on page 145 for additional information.</p>



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.
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 Section 3 - 4 "Ordered Data" for more information on ordered zones, and Section 3 - 5 "Finite-Element Data" for details on finite-element data. ZoneType, please note that some features in Tecplot 360 are limited by zone type. For example, iso-surfaces and slices are available for 3D zones types only (FETETRAHEDRON, FEBRICK, FEPOLYHEDRON and ORDERED - with K greater than 1). However, the plot type that you specify (in Tecplot 360 once you have loaded your data) is not limited by your zone type. You may have a 3D zone displayed in a 2D Cartesian plot (and visa versa).
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 (i.e. not ordered zones). Specify the total number of NODES , ELEMENTS and FACES in the data file. Refer to Section 3 - 5 "Finite-Element Data" for additional information.
ELEMENTS	= <integer>	Y		
FACES	= <integer>	Y		
TOTALNUMFACES	= <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.
NUMCONNECTEDBOUNDARYFACES	= <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.



Keyword	Syntax	Required (Y/N)	Default	Notes
TOTALNUMBERBOUNDARYCONNECTIONS	= <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.
FACENEIGHBORMODE	= [LOCALONE TOONE, LOCALONET OMANY, GLOBALON ETOONE, GLOBALON ETOMANY]	N	LOCALONE NETOONE	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 Section "Face Neighbor Connections List" on page 146 for details.
FACENEIGHBORCONNECTIONS	= <integer>	Y, if FACENEIGHBORMODE 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 Section "Face Neighbor Connections List" on page 146 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 DATAPACKING must be BLOCK. Refer to "Data" on page 142 for details.
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).



Keyword	Syntax	Required (Y/N)	Default	Notes
VARLOCATION	<code>=(<i>[set-of-vars]</i>=<varlocation>,<i>[set-of-vars]</i>=<varlocation>, ...)</code>	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.
VARSHARELIST	<code>=(<i>[set-of-vars]</i>=<zone>,<i>[set-of-vars]</i>=<zone>)</code>	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.
NV	<code>= <integer></code>	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 Section “Finite-Element Zone Node Variable Parameters Example” on page 179 for an example using the NV parameter.
CONNECTIVITYSHAREZONE	<code>=<zone></code>	N		Specify the number of the zone from which the connectivity is shared. The connectivity list (cell-based finite-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.
STRANDID	<code>= <integer></code>	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.



Keyword	Syntax	Required (Y/N)	Default	Notes
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 .
PARENTZONE	= <zone>	N		Scalar integer value representing the relationship between this zone and its parent. A value of zero indicates that this zone is not associated with a parent zone. A value greater than zero is considered this zone's parent. A zone may not specify itself as its own parent. With a parent zone association, Tecplot 360 can generate a surface streamtrace on a no-slip boundary zone. Refer to Section 16-3 "Surface streamtraces on no-slip boundaries" in the User's Manual for additional information.
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:

$$\begin{array}{lll} \mathbf{A}_{11} & \mathbf{A}_{12} & \dots \mathbf{A}_{1P} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \dots \mathbf{A}_{2P} \\ \cdot & & \\ \cdot & & \\ \cdot & & \\ \mathbf{A}_{V1} & \mathbf{A}_{V2} & \dots \mathbf{A}_{VP} \end{array}$$

where:

$$\begin{array}{l} \mathbf{V} = \text{total number of nonpassive, nonshared variables} \\ \mathbf{P} = \mathbf{I} * \mathbf{J} * \mathbf{K} \quad (\text{ordered zones}) \quad \text{or} \quad \mathbf{NODES} \quad (\text{FE zones}) \end{array}$$

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:

$$\begin{array}{lll} \mathbf{A}_{11} & \mathbf{A}_{12} & \dots \mathbf{A}_{1P} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \dots \mathbf{A}_{2P} \\ \cdot & & \\ \cdot & & \\ \cdot & & \\ \mathbf{A}_{V1} & \mathbf{A}_{V2} & \dots \mathbf{A}_{VP} \end{array}$$

where:

$$\begin{array}{l} \mathbf{V} = \text{total number of nonpassive, nonshared variables} \\ \mathbf{P} = (\mathbf{I} - 1) * (\mathbf{J} - 1) * (\mathbf{K} - 1) \quad (\text{ordered zones}^1) \\ \text{or} \\ \mathbf{P} = \mathbf{ELEMENTS} \quad (\text{FE zones}) \end{array}$$

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.

$$\begin{array}{lll}
 \mathbf{A}_{11} & \mathbf{A}_{12} & \dots \mathbf{A}_{1V} \\
 \mathbf{A}_{21} & \mathbf{A}_{22} & \dots \mathbf{A}_{2V} \\
 \cdot & & \\
 \cdot & & \\
 \cdot & & \\
 \mathbf{A}_{p1} & \mathbf{A}_{p2} & \dots \mathbf{A}_{pV}
 \end{array}$$

where:

\mathbf{V} = total number of nonpassive, nonshared variables

$\mathbf{P} = \mathbf{I} * \mathbf{J} * \mathbf{K}$ (ordered zones)

or

$\mathbf{P} = \mathbf{ELEMENTS}$ (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.
- 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, FEQUADILATERAL, 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](#).

Connectivity

For cell-based finite-element zones (FETRIANGLE, FEQUADILATERAL, 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”](#) on page 150.

The connectivity for face-based zones (FEPOLYGON and FEPOLYHEDRAL) is defined by the [Facemap Data](#) (refer to [“Facemap Data”](#) on page 148 for details).

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 **FACE-NEIGHBORMODE**, 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, zrn, crn, zr1, cr1, 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 *n*th 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 *n*th neighboring cell in the **GLOBALONETOMANY** list
- **crn** - the cell number in the remote zone of the *n*th neighboring cell in the **GLOBALONETOMANY** list.



The **cz**, **fz** combinations must be unique; multiple entries are not allowed. The face numbers for cells in the various zone types are defined in [Figure 3-1](#).

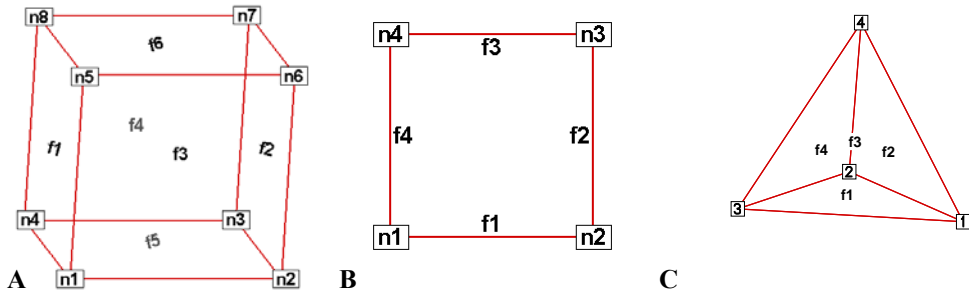


Figure 3-1. A: Example of node and face neighbors for an fe-brick cell or IJK-ordered cell. B: Example of node and face numbering for an IJ-ordered cell. C: example of tetrahedron face neighbors.

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#)
```

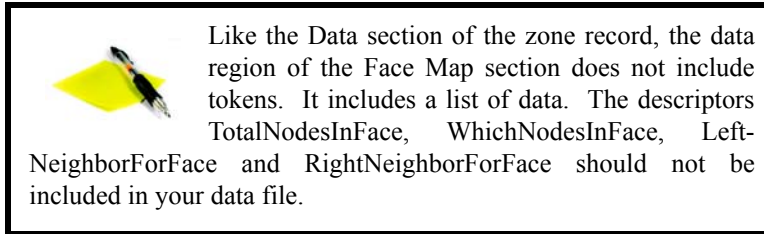
Global face neighbors are useful for telling Tecplot 360 about the connections between zones. This could be used, for example, to smooth out the crease in Gouraud surface shading at zone boundaries. For cell-centered data, they can make contours and streamtraces more continuous at zone boundaries.

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.



The face map data (in four major groupings) is defined by the following list:



1. **TotalNodesInFace** - A space-separated list of the total number of nodes in each face:

NodesInFace₁ NodesInFace₂ ...NodesInFace_F

where F is equal to the total number of faces.

NOTE: The TotalNodesInFace section is not used for polygonal zones, as each face of a polygon always has two nodes.

2. **WhichNodesInFace** - A list of the node numbers for each node in each face. Use a separate line for each face.

Face₁Node₁ Face₁Node₂ ...Face₁Node_{TotalNodesInFace1}

Face₂Node₁ Face₂Node₂ ...Face₂Node_{TotalNodesInFace2}

...

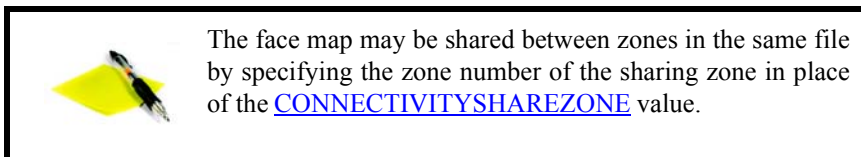
Face_FNode₁ Face_FNode₂ ...Face_FNode_{TotalNodesInFaceF}

3. **LeftNeighborForFace** - A list of left neighboring elements for each face:

LeftElementForFace₁ LeftElementForFace₂ ... LeftElementForFace_F

4. **RightNeighborForFace** -A list of right neighboring elements for each face:

RightElementForFace₁ RightElementForFace₂ ... RightElementForFace_F



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 tth boundary face. Refer to [Section "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"](#) on page 150.

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 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. 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.

3- 3.3 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 (<i>zone</i>) parameter to attach text to a specific zone or XY mapping. For further information, see Section 23-1.2 "Text Options" in 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 .
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 <i>Show in "Like" Frames</i> 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		Box Fill Color; BX (boxtype) must be set to FILLED .
BXM	= <double>	N		When BX (boxtype) is set to HOLLOW or FILLED , use the BXM token to specify the margin around text in box as fraction of H (<i>text height</i>).
BXO	= <color>	N		When BX (boxtype) is set to HOLLOW or FILLED , use the BXO token to specify the color of the box outline.



Token	Syntax	Required (Y/N)	Default	Notes
LT	= <double>			When BX (boxtype) is set to HOLLOW or FILLED , use the LT token to specify the thickness of the box outline.
F	= 	N		Use the F parameter to specify the font family.
C	= <color>	N		Font color.
AN	= <textanchor>			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 3-2 .
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.
H	= <double>			Specify the height, measured in the units defined by HU .
HU	= <heightunits>			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>			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 <i>tecplot.mcr</i> file. Refer to Section 23 - 5 "Text and Geometry Links to Macros" in the <i>User's Manual</i> and "\$!MACROFUNCTION...\$!ENDMACROFUNCTION" in the <i>Scripting Guide</i> for additional information.
CLIPPING	= <clipping>			Plot the geometry within to the viewport or the frame.

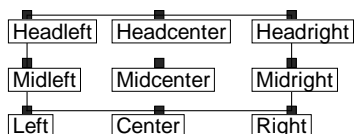


Figure 3-2. Text anchor positions—values for the [AN](#) parameter.



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"
```

3- 3.4 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 text to a specific zone or XY mapping. For further information, see Section 23- 1.2 "Text Options" in 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 is for LINE3D geometries only and must always be in GRID units. Refer to Section "Origin positions" on page 155 for additional information regarding the origin location for each type of geometry.
Y	= <double>	
Z	= <double>	
R	= <double>	Specify the r-origin and theta-origin of the object. The origins should be in CS units. Refer to Section "Origin positions" on page 155 for additional information.
THETA	= <double>	



Token	Available Values	Notes
CS	= <coordinatesys >	Geometry 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 (<i>scope</i>) 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 <i>Show in "Like" Frames</i> in the Geometry Options dialog.
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	= <arrowheadstyl e>	Arrowhead style
AAT	= N<arrowheadatt ach>	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 <i>tecplot.mcr</i> file. Refer to Section 23 - 5 "Text and Geometry Links to Macros" in the <i>User's Manual</i> and " \$!MACROFUNCTION...\$!ENDMACROFUNCTION " on page 185 in the 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

```

3- 3.5 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 [Section 18- 6.1 “Using Custom Labels”](#) in 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 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.



3- 3.6 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 [Section 3- 3.7 “Variable Auxiliary Data Record”](#). Auxiliary data for a given zone is defined by the [AUXDATA](#) token within the zone record (refer to [“Zone Header”](#) on page 138 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.



3- 3.7 Variable Auxiliary Data Record

Variable auxiliary data is added to Tecplot 360 on a per variable basis. Like dataset auxiliary data, multiple items can be added for each variable:

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

The variable number with which the auxiliary data is associated immediately follows the VARAUXDATA record. Also note that the data associated with a particular auxiliary data name are unique for each variable. Therefore the same named item can be added to each variable if desired. Conversely a particular auxiliary data item can be added to only one variable. NOTE: The name of an auxiliary data record cannot contain spaces.

3- 3.8 ASCII Data File Parameter Assignment Values

The following parameters 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
<draworder>	AFTERDATA, BEFOREDATA
	HELV, HELV-BOLD, TIMES, TIMES-ITALIC, TIMES-BOLD, TIMES-ITALIC-BOLD, COURIER, COURIER-BOLD, GREEK, MATH, USER-DEF
<geomtype>	LINE, SQUARE, RECTANGLE, CIRCLE, ELLIPSE
<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



<code><textanchor></code>	LEFT, CENTER, RIGHT, MIDDLEFT, MIDCENTER, MIDRIGHT, HEADLEFT, HEADCENTER, HEADRIGHT
<code><varlocation></code>	NODAL, CELLCENTERED
<code><zone></code>	zone number to which this item is assigned (0=all)
<code><zonetyp></code>	ORDERED, FELINESEG, FETRIANGLE, FEQUADRILATERAL, FETETRAHEDRON, FEBRICK, FEPOLYGON or FEPOLYHEDRAL

3 - 4 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.

3- 4.1 I-Ordered Data

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

3- 4.2 IJ-Ordered Data

IJ-ordered data has two indices: I and J. IJ-ordered data is typically used for 2D and 3D surface mesh, contour, vector, and shade plots, but it can also be used to plot families of lines in XY-plots. Refer to [Chapter 3 “Data Structure”](#) in 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 $JMax=1$.

3- 4.3 IJK-Ordered Data

IJK-ordered data has three indices: I, J, and K. This type of data is typically used for 3D volume plots, although planes of the data can be used for 2D and 3D surface plots. See [Section 3 - 1 “Ordered Data”](#) in 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 $KMax=1$.

3- 4.4 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 3-3](#); each data point is labeled with its I-index.

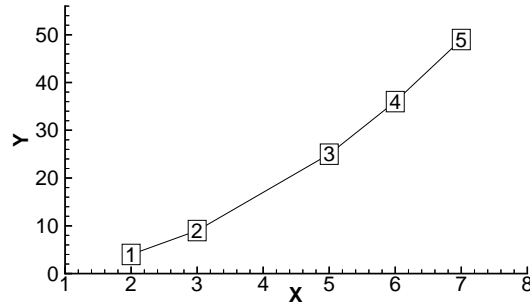


Figure 3-3. An I-ordered data set.

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.

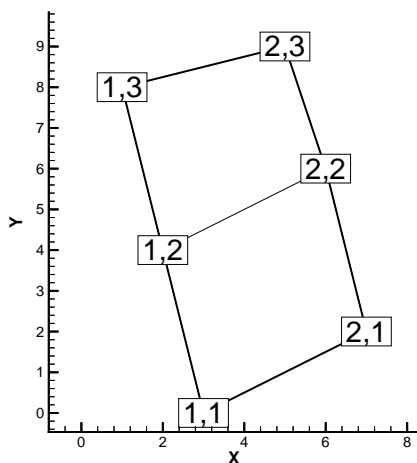


Figure 3-4. An IJ-ordered data set.

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 3-5](#); each data point is labeled with its IJK-index.

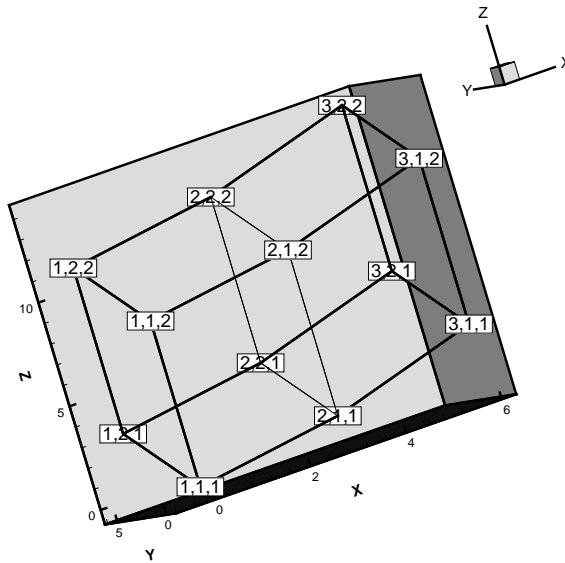


Figure 3-5. An IJK-ordered data set.

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

Time = 0.1 seconds:		
Position	Temperature	Pressure
71.31	564.9	101362.1
84.42	553.1	101348.9
103.1	540.5	101344.0
124.8	458.5	101342.2

For this case, we want to set up two zones in the data file, one for each time value. Each zone has three variables (**Position**, **Temperature**, and **Pressure**) and four data points (one for each location). This means that $IMax=4$ for each zone. We include a text record (discussed in [Section 3-](#)



[3.3 “Text Record”](#)) to add a title to the plot. The plot shown in [Figure 3-6](#) can be produced from this file.

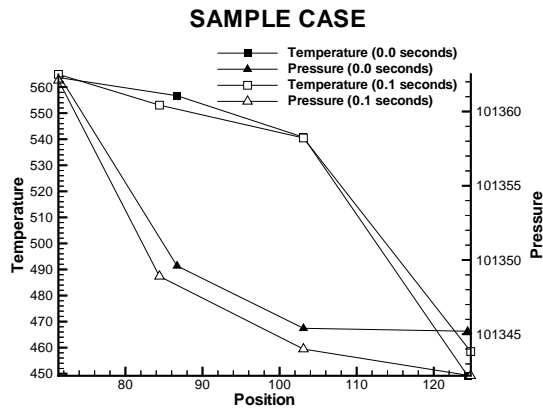


Figure 3-6. A multi-zone XY Line plot.

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.

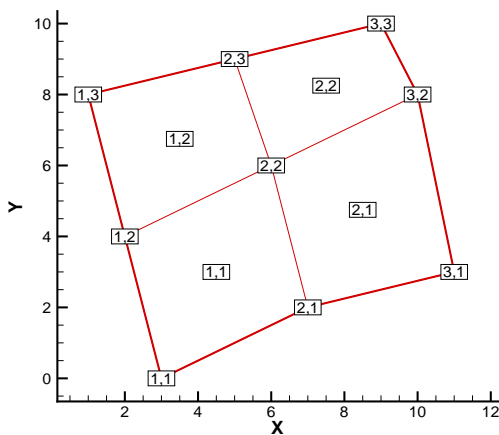


Figure 3-7. An IJ-ordered data set with cell-centered data.

```

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 Sidebar.

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

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

This data file has two zones and five variables, and is included with Tecplot 360 as the file **examples/dat/multzn2d.dat**. The first zone has nine data points arranged in a three-by-three grid (I=3, J=3). Each row of each zone represents one data point, where each column corresponds to the value of each variable for a given data point, i.e. X = 1.0, Y = 2.0, Press = 100.0, Temp = 50.0, and Vel = 1.0 for data point 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 [Figure 3-13](#).



Refer to [Section “Two-Dimensional Field Plots”](#) on page 177 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  
...
```

The complete ASCII data file is included with Tecplot 360 as **simp3dpt.dat** (POINT format), and in block format as **simp3dbk.dat**. When you read either of these files into Tecplot 360, the plot will appear as shown in [Figure 3-8](#).

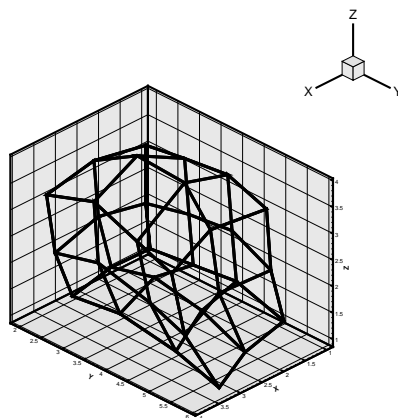


Figure 3-8. Plot of a 3D volume.



3 - 5 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).
- **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 3-9](#) shows the basic brick connectivity. For example, to create a tetrahedron, you can set $\mathbf{N3_M=N4_M}$ and $\mathbf{N5_M=N6_M=N7_M=N8_M}$. To create a quadrilateral-based pyramid, you can set $\mathbf{N5_M=N6_M=N7_M=N8_M}$.

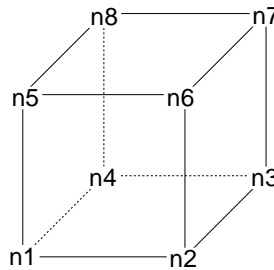


Figure 3-9. Basic brick connectivity.

- **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).

Refer to [Section 3-3.2 “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 [Section “Data”](#) on page 142 for details on arranging the data. The information following the nodal data is dependent upon the zone type.

For cell-based zone types (**FETRIANGLE**, **FEQUADILATERAL**, **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, **ZONE-TYPE=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 [Section “Connectivity”](#) on page 145 for details. You may also define Face Neighbors following the connectivity list. Refer to [Section “Face Neighbor Connections List”](#) on page 146 for details.

For face-based zone type (**FEPOLYGON** and **FEPOLYHEDRAL**), the data section ([Section “Data”](#) on page 142) is followed by the zone footer and facemap data sections. Refer to [Section “Facemap Data”](#) on page 148 for details.



3- 5.1 Variable and Connectivity List Sharing

The **VARSHARELIST** in the **ZONE** record allows you to share variables from specified previous zones. The **CONNECTIVITYSHAREZONE** parameter in the **ZONE** record allows you to share the connectivity list from a specified previous zone. The following is an example to illustrate these features. 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 (P_1 , P_2 , P_3). The XY locations have been arranged into finite-elements.

X	Y	P ₁	P ₂	P ₃
-1.0	0.0	100	110	120
0.0	0.0	125	135	145
1.0	0.0	150	160	180
-0.5	0.8	150	165	175
0.5	0.8	175	185	195
0.0	1.6	200	200	200

For this case, we want to set up three zones in the data file, one for each time measurement. Each zone has three variables: X, Y, and P. The zones are of the triangle element type, meaning that three nodes must be used to define each element. One way to set up this data file would be to list the complete set of values for X, Y, and P for each zone. Since the XY-coordinates are exactly the same for all three zones, a more compact data file can be made by using the **VARSHARELIST**. In the data file given below, the second and third zones have variable sharing lists that share the values of the X- and Y-variables and the connectivity list from the first zone. As a result, the only values listed for the second and third zones are the pressure variable values. Note that the data could easily have



been organized in a single zone with five variables. Since blank lines are ignored in the data file, you can embed them to improve readability. A plot of the data is shown in [Figure 3-10](#).

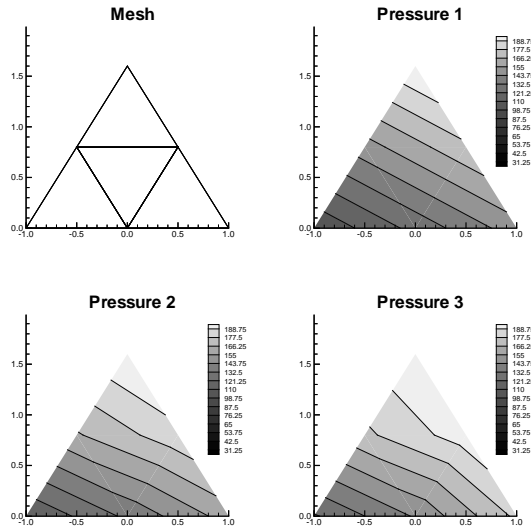


Figure 3-10. A plot of finite-element zones.

```

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

```

3- 5.2 Finite-Element Data Set Examples

Creating a finite-element data set is generally more complicated than creating a similar-sized ordered data set¹. In addition to specifying all the data points, you must also specify the connectivity list. Consider the data shown in [Table 3 - 2](#).

Table 3 - 2: finite-element Data

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

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 3-11](#)):

```

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

```

1. Background information for FE data sets is provided in [Section 3 - 2 "Finite-Element Data"](#) in the [User's Manual](#).



```

2.0 2.0 280.0 1.9
1 2 5 4
2 3 6 5
6 7 3 3
3 2 8 8

```

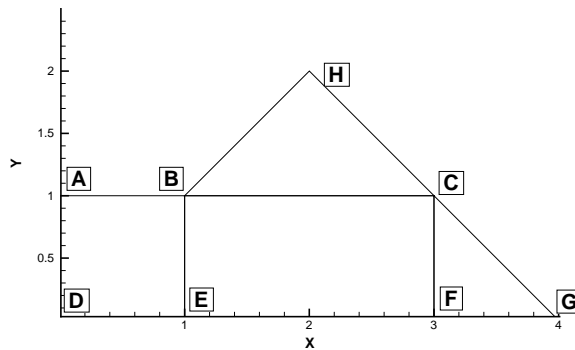


Figure 3-11. A mesh plot of 2D finite-element data.

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 3-12](#). (You must also change the **ELEMENTS** parameter in the zone control line to specify five elements.)

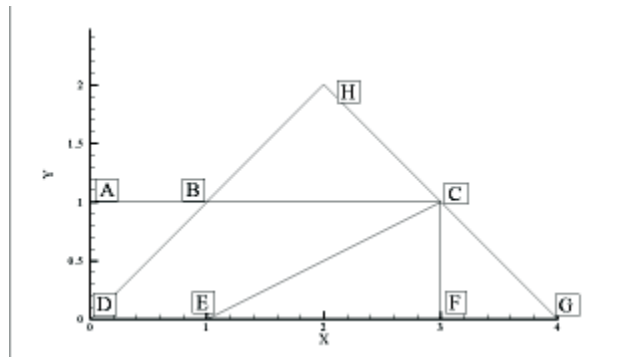


Figure 3-12. Finite-element data of [Figure 3-11](#) with a different connectivity list

```

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 (included in your Tecplot 360 distribution as **examples/dat/2dfed.dat**):

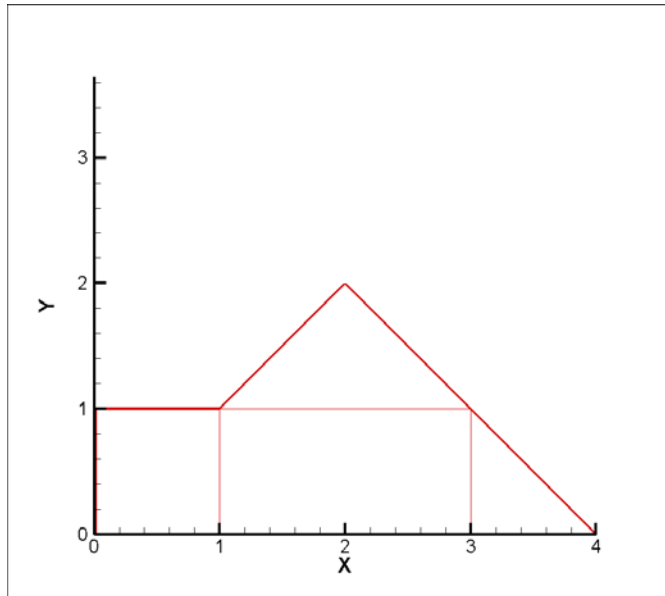


Figure 3-13. A 2D field plot.

```
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 [Chapter 3 “Data Structure”](#) in 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 3-14](#). Each data point is labeled with its node number.

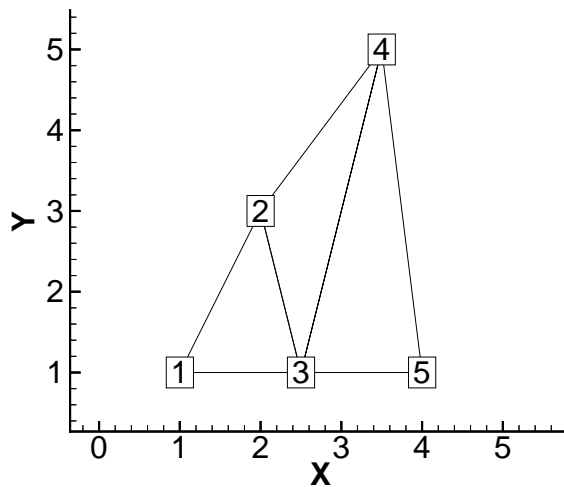


Figure 3-14. A finite-element triangle data set.

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 3 -3](#). 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).

Table 3 - 3: Data set with eleven nodes and three variables.

X	Y	Z
0.0	0.0	1.0
0.0	0.0	-2.0



Table 3 - 3: Data set with eleven nodes and three variables.

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

A Tecplot 360 data file for the data in [Table 3 - 3](#) is shown below in **POINT** format and plotted in [Figure 3-15](#):

```

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

```

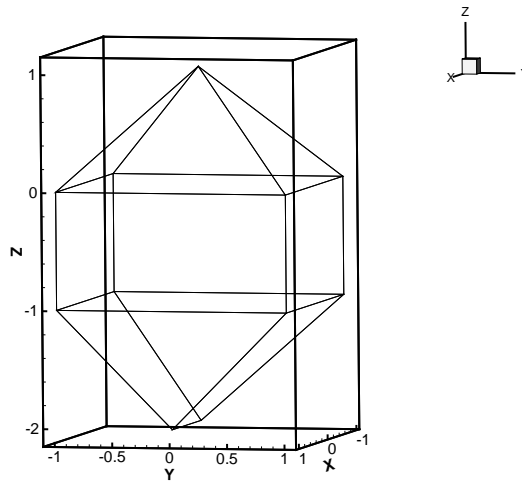


Figure 3-15. Three-dimensional mesh plot of finite-element surface data.

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 3 - 4](#). The data can be divided into five brick elements, each of which is defined by eight nodes.

Table 3 - 4: Finite-Element Volume - Brick Data Set. Data with 14 nodes and four variables.

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

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 3-9](#) 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 included in your distribution (*examples/dat/febrfef.dat*) and 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 included in your distribution (*examples/dat/febrfeb.dat*) and 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 13 102 2 4 4 7 6 9 10
```



[Figure 3-16](#) shows the resulting mesh plot from the data set listed in this section.

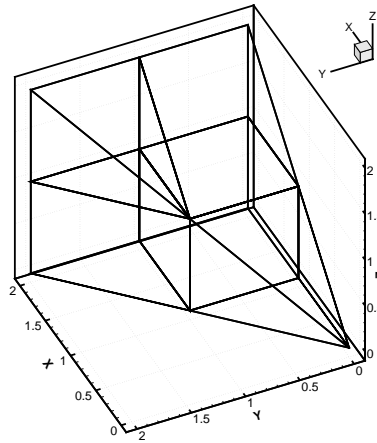


Figure 3-16. A finite-element brick zone.

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 3 - 5](#). 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.

Table 3 - 5: Finite-Element Volume - Tetrahedral data set with 13 nodes and seven variables.

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



Table 3 - 5: Finite-Element Volume - Tetrahedral data set with 13 nodes and seven variables.

X	Y	Z	C	U	V	W
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

The data file in **POINT** format for the data in [Table 3 - 5](#) is shown below, and plotted in [Figure 3-17](#):

```

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

```

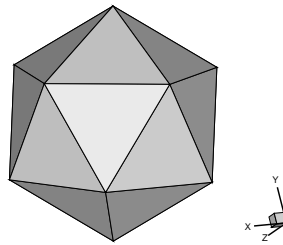
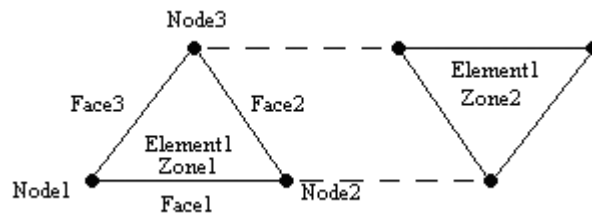


Figure 3-17. Finite-element volume tetrahedral data.

This data file is included in your Tecplot 360 distribution's `examples/dat` directory as the file `fetetpt.dat`. A block format version of the same data is included as the file `fetetbk.dat`.

Polygonal - simple example

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

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

```

3 - 6 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. Your Tecplot 360 distribution includes Preplot, which converts ASCII to binary data files. You can also use Preplot to debug ASCII data files that Tecplot 360 cannot read.

3- 6.1 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 either the standard set of Preplot options or from a special set of options for reading PLOT3D format files. If *outfile* is not specified, the binary data file has the same base name as the *infile* with a **.plt** extension. You may use a minus sign (“-”) in place of either the *infile* or *outfile* to specify standard input or standard output, respectively.

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

For more Preplot command lines, see [Section B - 4 “Preplot”](#) in the [User’s Manual](#).

3- 6.2 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.



The following terms are used throughout the *Data Format Guide* and are included here for your reference.

2D	<i>Plotting in two dimensions. Line plots of one or more variables (XY and Polar Line plots) are not considered 2D.</i>
2D Cartesian Plot	<i>A plot of some variable by location on a single plane using two axes.</i>
3D	<i>Plotting in three dimensions. Three-dimensional plotting can be subdivided into 3D surface and 3D volume.</i>
3D Cartesian Plot	<i>A plot displaying a 3D scattering of points, surfaces, or volumes using three orthogonal axes.</i>
3D Surface	<i>Three-dimensional plotting confined to a surface. For example, the surface of a wing.</i>
3D Volume	<i>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.</i>
Active Zone	<i>A zone that is displayed in the current plot, as determined in the Zone Style dialog.</i>
ASCII Data File	<i>A data file composed of human-readable statements and numbers using ASCII characters.</i>
Auxiliary Data	<i>Metadata attached to zones, data sets, and frames.</i>
Binary Data File	<i>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.</i>



Block	<i>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.</i>
Boundary Cell Faces	<i>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.</i>
Brick	<i>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.</i>
Cell	<i>Either an element of finite-element data, or the space contained by one increment of each index of IJ- or IJK-ordered data.</i>
Cell-Centered Values	<i>Values located at the center of the cell (assumed to be the centroid).</i>
Connectivity List	<i>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.</i>
Custom Labels	<i>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.</i>
Data File	<i>A file that contains data used for plotting in Tecplot.</i>
Data Format	<i>The type of zone data as specified by the format parameter in a Tecplot data file, such as: BLOCK or POINT.</i>
Data Loader	<i>A Tecplot add-on which allows you to read non-Tecplot data files.</i>
Data Point	<i>An XYZ-point at which field variables are defined.</i>



Data Set	<i>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.</i>
Element Type	<i>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.</i>
FE	<i>An abbreviation for finite-element, a common means of arranging data for calculations. (Often referred to as “unordered” or “unstructured”.)</i>
FE Surface	<i>A finite-element zone of the element type Triangle, Quadrilateral, Polygon. These zones are used for 2D and 3D surface plots.</i>
FE Volume	<i>A finite-element zone of the element type Tetrahedron, Brick, Polyhedron. These zones are used for 3D volume plots.</i>
Field Map	<i>A collection of zones for 2D and 3D field plots. A common style can be easily applied to all zones in the selection.</i>
Field Plot	<i>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.</i>
Finite-Element	<i>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>



I-Ordered	<i>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 or triangulating 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.)</i>
IJ-Ordered	<i>A type of data point ordering where the points are arranged in a 2D array used for 2D and 3D surface plotting.</i>
IJK-Blanking	<i>A feature to include or exclude portions of an IJK-ordered zone based on index ranges.</i>
IJK-Ordered	<i>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>
I-Plane	<i>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.</i>
Irregular Data	<i>Points which have no order, or at least no order which can be easily converted to IJ- or IJK-ordering.</i>
J-Plane	<i>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.</i>
K-Plane	<i>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.</i>
Macro	<i>A file containing a list of instructions, called macro commands, which can duplicate virtually any action performed in Tecplot.</i>
Macro Command	<i>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.</i>
Macro File	<i>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.</i>
Macro Function	<i>A self-contained macro sub-routine.</i>



Macro Variable	<i>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).</i>
No Neighboring Element	<i>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.</i>
Node	<i>A point in finite-element data.</i>
Number Format	<i>The style of numbers to display for a data or axis label; exponent, integer, float, and so forth.</i>
Ordered Data	<i>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.</i>
Polygonal	<i>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.</i>
Polyhedral	<i>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.</i>
Point	<i>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.</i>
Quadrilateral	<i>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.</i>
Sharing	<i>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.</i>
Tetrahedron	<i>An element type of finite-element volume data which is composed of four node points arranged in a tetrahedron. (Used in 3D volume plotting.)</i>



Triangle	<i>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.)</i>
Unordered or Unorganized Data	<i>(See Irregular Data.)</i>
Zone	<i>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.</i>
Zone Layers	<i>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.</i>



Appendix A *Binary Data File Format*

Refer to this section only if you wish to write your own functions. Otherwise, refer to [Section 2 - 1 "Getting Started"](#) for instructions for linking with the library provided by Tecplot, Inc.

```
/*  
BINARY FILE FORMAT:  
-----  
The binary data file format (as produced by the preplot) is described below.
```

The binary datafile has 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

```
+-----+
| "#!TDV111"|      8 Bytes, exact characters "#!TDV111".
+-----+      Version number follows the "V" and
                    consumes the next 3 characters (for
                    example: "V75 ", "V101").
```

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 (NumVar) in the datafile.
+-----+

+-----+
| INT32*N    |      Variable names.
+-----+                    N = L[1] + L[2] + ... L[NumVar]
                    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
+-----+

+-----+
| INT32*N    |      Zone name. (See note 1.)
+-----+                    N = (length of zone name) + 1.
+-----+
```



```

| INT32      |      ParentZone: Zero-based zone number within this
+-----+      datafile to which this zone is
                        a child.

+-----+
| INT32      |      StrandID: -2 = pending strand ID for assignment
+-----+                        by Tecplot
                        -1 = static strand ID
                        0 <= N < 32700 valid strand ID

+-----+
| FLOAT64    |      Solution time.
+-----+

+-----+
| INT32      |      Zone Color (set to -1 if you want Tecplot to
+-----+                        determine).

+-----+
| INT32      |      ZoneType 0=ORDERED,          1=FELINESEG,
+-----+                        2=FETRIANGLE,       3=FEQUADRILATERAL,
                        4=FETETRAHEDRON, 5=FEBRICK,
                        6=FEPOLYGON,       7=FEPOLYHEDRON

+-----+
| INT32      |      DataPacking 0=Block, 1=Point
+-----+      FEPOLYGON and FEPOLYHEDRON zones require Block
                        formatting.

+-----+
| INT32      |      Specify Var Location.
+-----+      0 = Don't specify, all data is located
                        at the nodes.
                        1 = Specify

if "specify var location" == 1
+-----+
| INT32*N*V  |      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 | NumFaces
+-----+
+-----+
| INT32 | Total number of face nodes. 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,



```

+-----+
+-----+
| INT32*N | 1=ClipToViewport, 2=ClipToFrame)
+-----+
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
+-----+
+-----+
| 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*N |          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

```
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
+-----+
| INT32*F   |      Face node offsets into the face nodes array
+-----+      below. Does not exist for FEPOLYGON zones.
                F = NumFaces+1.

+-----+
| 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 connection elements array and
the boundary face connection zones array.
The number of elements for a face (F) is
determined by $\text{offset}[-o] - \text{offset}[-o-1]$
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.

+-----+
| INT16*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)

- 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
- 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 $I_{Max} \times J_{Max} \times K_{Max}$ numbers of cell centered values, where I_{Max} , J_{Max} , and K_{Max} 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 slowest moving indices. For example, if your data's IJK dimensions are $2 \times 3 \times 2$, a cell-centered variable will have $1 \times 2 \times 1$ (i.e. $(I-1) \times (J-1) \times (K-1)$) significant values. However, $2 \times 3 \times 2$ 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 $2 \times 3 \times 2$ its cell centered variable would be represented as follows:

```
1.5  0.0  12.5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

If the zone was dimensioned $3 \times 2 \times 2$ its cell centered variable would be represented as follows:

```
1.5  12.5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

and if the zone was dimensioned $2 \times 2 \times 3$ 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  0.0  0.0  0.0  0.0
```

For large variables the wasted space is less significant than it is for the small example above.

*/



Index

A

- Anchor
 - text position 152
- ASCII Data
 - conversion to binary 133, 189
 - Custom Label Record 157
 - File Format 134–160
 - finite-element data 170–189
 - Geometry Record 153–157
 - ordered data 160–169
 - parameters 159
 - syntax 144
 - Text Record 150–153
 - Zone Record 137–150
- ASCII format
 - syntax 133
- Auxiliary Data 14, 141
 - variable auxiliary data 39, 159
 - zone auxiliary data 41
- Axis Labels 30

B

- Binary Data
 - byte order 22
 - conversion from ASCII 133, 189
 - File Format 197–212
 - geometry creation 23
 - text record 35
 - user record 39
- Binary files
 - debugging 10
 - writing to multiple 12, 21
 - writing to multiple, example 121
- Block Data 143
- Boundary Connection 48
- Boundary Face 48
- Boundary Map 150
- Brick cells 170
- Byte order 22

C

- Cell-centered Data 18, 143
- Connected Boundary Face 48
- Connectivity list 145
 - cell-based finite elements 31
 - face-based finite elements 32

- sharing 140, 150
- Custom Label Record
 - ASCII data 157
 - binary data 30

D

- Data Arrangement 17
- Data conversion 133, 189
- Data File Format
 - ASCII 134–160
 - binary 197–212
- Data Types 142

E

- EOF 19
- Examples
 - ASCII
 - auxiliary data 158
 - finite-element 174–189
 - Geometry 156
 - ordered data 161–169
 - Text Record 153
 - Binary
 - Face Neighbors 54
 - geometry record 128
 - IJ-ordered 118
 - polygonal data 65
 - polyhedral zones 73, 92, 113
 - text record 126

F

- Face Neighbors 146, 150
 - data 19
 - example 54
 - mode 147
 - polyhedral zones 52
 - right-hand rule 52
 - scope 147
- Face Numbering
 - cell-based finite elements 148
- Facemap data 32, 148
 - polyhedral zones 50
- File
 - grid file 30
 - shared grid 30
 - solution file 30
- File Format
 - ASCII 134–160
 - Binary Data 197–212
- File Header 135



Finite-element data
 ASCII format 170–189
 boundary map 150
 bricks 170
 connectivity list 31, 32, 145
 face neighbors 19
 face numbering (cell-based) 148
 facemap 148
 line segments 170
 polygons 170
 polyhedra 171
 polyhedral format 32
 quadrilaterals 170
 tetrahedron 170
 triangles 170
 Full file 30
 Function reference
 TecIO library 14–48
 Function sequence
 binary files 12

G

Geometry Record
 ASCII data 153–157
 binary
 example 128
 syntax 23
 data (ASCII) 155
 origin positions 28
 Global one-to-many 147
 Global one-to-one 147
 Grid
 sharing 30
 Grid File 30

H

Header
 file header 135
 zone header 42

I

Irregular data 160

L

Labels, custom
 binary data 30
 Legend text 30
 Line Segments 170
 Local one-to-many 147
 Local one-to-one 147

M

Metadata, *see Auxiliary Data*

N

Neighboring elements 150
 Nodal Data 17, 143

O

Ordered Data 160–169
 Example (binary) 118
 Examples
 2D Field Plot 168
 3D Field Plot 169
 IJK-ordered 164
 IJ-ordered 163
 I-ordered 162
 Examples (ASCII) 161–169
 IJK-ordered data 161
 IJ-ordered data 160
 I-ordered data 160
 one-dimensional 160
 three-dimensional 161
 two-dimensional 160
 Origin positions
 geometry 28

P

Parameters
 ASCII data file 159
 Pltview 10
 Polygonal zones 170
 Polyhedral cells 171
 Polyhedral data
 boundary connection 48
 boundary face 48
 Examples (binary)
 multiple zones (2D) 92
 multiple zones (3D) 73
 polygon 65
 polyhedral 113
 face neighbors 52
 facemap data 50
 Preplot 133, 189

Q

Quadrilateral cells 170

R

Right-hand rule



face neighbors 52

S

Scatter Plots 160

Shared grid 30

Solution file 30

Syntax

ASCII format 133

TecIO functions 14–48

T

TECAUXSTR111 14

TECDAT111 15

TECEND111 19

TECFACE 19

TECFIL 21

TECFOREIGN 22

TECGEO 23

TecIO functions 14–48

TecIO library 9

depreciated functions 11

function calling sequence 12

function reference 14–48

linking with 13

TECLAB 30

TECNOD 31

TECPOLY 32

TECTXT 35

TECUSR 39

TECVAUXSTR 39

TECZAXSTR 41

TECZNE 42

Tetrahedral cells 170

Text Anchor 152

Text Record

ASCII data 150–153

Binary Data 35

example 126

Text Anchor positions 152

Tick mark Labels 30

Triangular Cells 170

Triangulation 160

U

Unstructured Data 160

User record

binary data 39

V

Variable auxiliary data 39

Variable Location 140, 142–144

Variable Sharing 140, 144, 172

ViewBinary 10

X

XY Plot

example 165

XY Plots 160

Z

Zone auxiliary data 41

Zone Footer 145

Zone header 42

Zone Record 137–150

Zone Type

finite-element zones 170

Zone Types 138, 160

FEBRICK 170

FELINESEG 170

FEPOLYGON 170

FEPOLYHEDRAL 171

FEQUADRILATERAL 170

FETETRAHEDRON 170

FETRIANGLE 170



