



User's Manual

*Add-On Developer's Kit
for Tecplot 360 2013 R1*

COPYRIGHT NOTICE

Tecplot 360™ User's Manual is for use with Tecplot 360™ Version 2013 R1.

Copyright © 1988-2013 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

LAPACK 1992-2007 LAPACK Copyright © 1992-2007 The University of Tennessee. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this sense in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The University of Tennessee. All Rights Reserved. SciPy 2001-2009 Enthought, Inc. All Rights Reserved. NumPy 2005 NumPy Developers. All Rights Reserved. VisTools and VdmTools 1992-2009 Visual Kinematics, Inc. All Rights Reserved. NCSA HDF & HDF5 (Hierarchical Data Format) Software Library and Utilities Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois, Fortner Software, Unidata Program Center (netCDF), The Independent JPEG Group (JPEG), Jean-loup Gailly and Mark Adler (gzip), and Digital Equipment Corporation (DEC). Conditions of Redistribution: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution. 3. In addition, redistributions of modified forms of the source or binary code must carry prominent notices stating that the original code was changed and the date of the change. 4. All publications or advertising materials mentioning features or use of this software are asked, but not required, to acknowledge that it was developed by The HDF Group and by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign and credit the contributors. 5. Neither the name of The HDF Group, the name of the University, nor the name of any Contributor may be used to endorse or promote products derived from this software without specific prior written permission from the University, THG, or the Contributor, respectively. DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP (THG) AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. In no event shall THG or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage. Copyright © 1998-2006 The Board of Trustees of the University of Illinois, Copyright © 2006-2008 The HDF Group (THG). All Rights Reserved. PNG Reference Library Copyright © 1995, 1996 Guy Eric Schalnat, Group 42, Inc., Copyright © 1996, 1997 Andreas Dilger, Copyright © 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 Scripps Corporation. All Rights Reserved. bmtopnm 1992 David W. Sanderson. All Rights Reserved. Netpbm 1988 Jef Poskanzer. All Rights Reserved. Mesa 1999-2003 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. Pmtopic 1990 Ken Yap. All Rights Reserved. JPEG 1991-1998 Thomas G. Lane. All Rights Reserved. Diredt API for Microsoft Visual Studio (diredt.h) 2006-2006 Copyright © 2006 Toni Ronkko. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so. Toni Ronkko. All Rights Reserved. ICU 1995-2009 Copyright © 1995-2009 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. International Business Machines Corporation and others. All Rights Reserved. QsLog 2010 Copyright © 2010, Razvan Petru. All rights reserved. QsLog Copyright (c) 2010, Razvan Petru. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. The name of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Razvan Petru. All Rights Reserved. VTK 1993-2008 Copyright © 1993-2008 Ken Martin, Will Schroeder, Bill Lorensen. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither name of Ken Martin, Will Schroeder, or Bill Lorensen nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Ken Martin, Will Schroeder, Bill Lorensen. All Rights Reserved.

TRADEMARKS

Tecplot,® Tecplot 360,™ the Tecplot 360 logo, Preplot,™ Enjoy the View,™ Master 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. EnSight is a registered trademark or trademark of Computation Engineering International (CEI), Incorporated in the U.S. and/or other countries. EDEM is a registered trademark or trademark of DEM Solutions Ltd in the U.S. and/or other countries. Exceed 3D, Hummingbird, and Exceed are registered trademarks or trademarks 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 Halliburton in the U.S. and/or other countries. ECLIPSE FrontSim is a registered trademark or trademark of Schlumberger Information Solutions (SIS) in the U.S. and/or other countries. Debian is a registered trademark or trademark of Software in the Public Interest, Incorporated 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 Window System 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 are registered trademarks or trademarks of ANSYS Incorporated or its subsidiaries in the U.S. and/or other countries. PAM-CRASH is a registered trademark or trademark of ESI Group in the U.S. and/or other countries. LS-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. 3DSL is a registered trademark or trademark of StreamSim Technologies, Incorporated 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. Star-CCM+ is a registered trademark or trademark of CD-adapco in the U.S. and/or other countries. Reprise License Manager is a registered trademark or trademark of Reprise Software, Inc. 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. Abaqus, the 3DS logo, SIMULIA and CATIA are registered trademarks or trademarks of Dassault Systèmes or its subsidiaries in the U.S. and/or other countries. The Abaqus runtime libraries are a product of Dassault Systèmes Simulia Corp., Providence, RI, USA. © Dassault Systèmes, 2007 FLOW-3D 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 are registered trademarks or trademarks of Adobe Systems, Incorporated in the U.S. and/or other countries. AutoCAD and DXF are registered trademarks or trademarks 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 are registered trademarks or trademarks of Hewlett-Packard Development Company, Limited Partnership in the U.S. and/or other countries. IBM, RS/6000 and AIX are registered trademarks or trademarks of International Business Machines Corporation in the U.S. and/or other countries. Helvetica Font Family and Times Font Family are registered trademarks or trademarks 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 are registered trademarks or trademarks of Microsoft Corporation in the U.S. and/or other countries. Firefox is a registered trademark or trademark of The Mozilla Foundation 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. Products bearing SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc. Solaris, Sun and SunRaster are registered trademarks or trademarks of Sun Microsystems, Incorporated in the U.S. and/or other countries. Courier is a registered trademark or trademark of Monotype Imaging Incorporated in the U.S. and/or other countries. UNIX and Motif are registered trademarks or trademarks 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. JPEG is a registered trademark or trademark of Thomas G. Lane in the U.S. and/or other countries. SENSOR is a registered trademark or trademark of Coats Engineering in the U.S. and/or other countries. SENSOR is licensed and distributed only by Coats Engineering and by JOA Oil and Gas, a world-wide authorized reseller. MySQL is a registered trademark or trademark of Oracle in the U.S. and/or other countries. MySQL is a trademark of Oracle Corporation and/or its affiliates. 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.

13-360-06-1

Rev 1/2013

Table Of Contents

<i>List of Tutorials</i>	11
--------------------------	----

Introduction

1	<i>Introduction</i>	19
	Platform Issues	20
	Add-ons Created with Previous Versions	20
	Recent Updates	20
	Macros Versus Add-ons	21
2	<i>Creating Add-ons on Linux/Macintosh Platforms</i>	23
	General Setup for Creating Add-ons	23
	Creating a New Add-on	24
	Compiling the Add-on	25
	<i>Using Runmake</i>	25
	<i>Editing the CustomMake File</i>	25
	Debugging Shared Objects	27
3	<i>Creating Add-ons on Windows Platforms</i>	29
	General Steps for Creating an Add-on	29
	Creating an Add-on with Visual Studio 2005	30
	<i>Creating an Add-on Using Visual Studio</i>	30
	<i>Debugging Your Add-on</i>	31
	Creating an Add-on with Intel Visual Fortran	34
	<i>Creating an Add-on Using Visual Fortran</i>	34
	<i>Debugging Your Add-on</i>	35

4	<i>Running Tecplot 360 with Add-ons</i>	39
	Specifying which Add-ons to Load.....	39
	<i>Add-ons Loaded by All Users</i>	39
	<i>Specifying a Secondary Add-on Load File</i>	39
	<i>Specifying Add-ons Via the Command Line</i>	39
	Using the \$!LoadAddOn Command	40
	Specifying Add-ons under Development	40
	<i>Developing Add-ons on Linux or Macintosh platforms</i>	40
	<i>Developing Add-ons on a Windows Operating System</i>	41

General Add-on Components

5	<i>Add-on Initialization and Cleanup</i>	45
	Add-on Initialization	45
	Add-on Cleanup	46
6	<i>State Changes From an Add-on</i>	47
	State Change Values	47
	Listening for State Changes	50
	<i>Supplemental Information</i>	52
	<i>Coding Rules for State Change Listeners</i>	54
	Sending State Changes	54
7	<i>Locking and Unlocking Tecplot 360</i>	57
8	<i>Working with Multi-threading</i>	59
	Thread pool	59
	Thread Mutex	60
	Condition Variables	60

Data

9	<i>Data Structure</i>	63
	Ordered Data	63
	<i>Face Neighbors for Ordered Data</i>	64
	Finite Element Data	65
	<i>Indexing Nodal Finite Element Data</i>	66
	<i>Indexing Cell-centered Finite Element Data</i>	66
	Face Neighbors	71
10	<i>Accessing Field Data</i>	75
	Data Access Methods	75

	By Reference (Array)	75
	By Reference (Single Value).....	76
	By Specifying Zone-variable	78
	Working with Shared Data.....	78
	Branching Shared Data	78
	Querying or Modifying Shared Data	79
	Allowing Data Sharing	79
	Load-on-demand	79
11	Manipulating Data	87
	Calculating Equations.....	87
	Equation Syntax	88
	Data Smoothing	94
	Limitations to Smoothing	94
	Coordinate Transformation	95
	Zone Creation.....	95
	1D Line Creation	96
	Rectangular Zone Creation	96
	Circular or Cylindrical Zone Creation.....	97
	Zone Duplication.....	98
	Mirror Zone Creation.....	99
	FE Surface Zone Creation (from Polylines)	100
	Zone Creation by Entering Values	100
	Data Extraction from an Existing Zone.....	100
	Sub-zone Extraction	101
	Data Interpolation.....	101
	Linear Interpolation.....	101
	Inverse-distance Interpolation.....	102
	Kriging Interpolation	104
	Irregular Data Point Triangulation	105
12	Auxiliary Data	107
	Understanding Auxiliary Data.....	107
	Using Standardized Auxiliary Data.....	108
13	Setting Plot Style	111
	Linemaps and Fieldmaps	111
	Linemap and Fieldmap Styles.....	115
	Line Plots.....	115
	Mapping Layers.....	117
	Line Plot Axes	119
	Curve Types	120
	Line Legends	121
	Field Plots	122
	Fieldmap Layers.....	122
	Plot Effects.....	134
	Derived Objects	135
	Plotting Subsets of Data	143
	Points	143

	<i>Surfaces</i>	144
	Coloring	146
	<i>Basic Colors</i>	147
	<i>Multi-color Settings</i>	147
	<i>RGB Multi-coloring</i>	148
	Axes	151
	<i>Axis Variable Assignment</i>	151
	<i>Axis Range</i>	152
	<i>Tick Marks</i>	152
	<i>Axis Lines</i>	153
	<i>Axis Labels</i>	153
	<i>Axis Grid Area</i>	153
	Annotations	153
	<i>Text</i>	153
	<i>Geometries</i>	162
	<i>Images</i>	164
	<i>Labels</i>	165
	Blanking	165
	Frame Linking	166
	Animation	169
	Working with Transient Data	170
	View Options	172
	<i>Rotation</i>	172
	<i>Translation</i>	173
	<i>Zoom</i>	173
14	<i>Code Generator</i>	175
	Style Hierarchy	175
	Code Generator.....	177
	<i>Launching the Code Generator</i>	177
	<i>Incorporating Code Generator Output</i>	177
	<i>Commands Not Displayed by the Code Generator</i>	178
15	<i>Porting Add-ons</i>	181
	Porting Add-ons from a Windows Platform to a Linux Platform	181
	Porting Add-ons from a Linux Platform to a Windows Platform	181

Lists and Sets

16	<i>Argument Lists</i>	185
	Tecplot Toolbox ArgList Class	185
	TecUtil Functions that Use Argument Lists	186
17	<i>String Lists</i>	187
	Tecplot Toolbox StringList Examples	187

18	<i>Sets</i>	191
	Tecplot Toolbox Set Class	191
	TecUtil Functions that Use Sets	193

Additional Functionality

19	<i>Augmenting the Macro Language of Tecplot 360</i>	199
	Processing Custom Macro Commands.....	199
	Error Processing.....	200
	Recording Custom Macro Commands.....	201
20	<i>Implementing Data Journaling</i>	205
	Data Journaling Prerequisites	206
	Inhibiting Marking of the Data Set.....	206
21	<i>Adding Online Help to Your Add-on</i>	209
	Step 1: Write your Help Pages.....	209
	Step 2: Create a Help Directory	209
	Step 3: Processing the Help Button Callback	209
	Using the Tecutilhelp Function.....	210
22	<i>Working with Picked Objects</i>	213
	Object Types.....	213
	Picking Objects.....	214
	<i>Picking Multiple Objects</i>	214
	Operating on Picked Objects	214
	Pick List	215

User Interface

23	<i>GUI Overview</i>	219
	UTF8 Strings	219
	Types of Dialogs	219
24	<i>Dialogs on Windows Platforms</i>	221
	Modal Dialogs	221
	Modeless Dialogs	222
	PreTranslateMessage Function for Modeless Dialogs	223

25	<i>Tecplot GUI Builder</i>	225
	Using Tecplot GUI Builder.....	225
	How TGB Works	225
	Selecting a Language	226
	Adding Dialogs and Sidebars	227
	Adding Dialogs	227
	Adding Sidebars	228
	Adding Controls.....	228
	Common Control Features	229
	Bitmap Buttons and Toggles	231
	Push Buttons	232
	Scale Controls	232
	Form Controls.....	232
	Tab Controls.....	233
	Text Field Spin Controls.....	234
	Option Menus	234
	Adding a Menu Bar to a Dialog.....	235
	Specifying Exact Control Width in Dialog Units or Characters	236
	Building the Source Code.....	237
	Modifying Your Source Code.....	237
	Compiling Your Add-on.....	238
	Linux and Macintosh OS X Platforms	238
	Windows Platforms	238
	Informing Tecplot 360 of Your New Add-on.....	238
	Running Your New Add-on	238

Common Add-ons

26	<i>Building Data Set Reader Add-ons</i>	245
	Converters Versus Loaders	245
	How a Data Converter Works	245
	How a Data Loader Works	245
27	<i>Creating a Data Loader</i>	247
	Data Hierarchy.....	248
	Primary Data Load Steps.....	250
	Recommended Code Layering	251
	Core Loader Function	252
	Immediate Loading	252
	Load-on-demand.....	255
	Auto Load-variable-on-demand.....	257
	Load Variable-on-Demand	260
	Load Value-on-demand	262
	Load-on-demand Callbacks and Threads	263
	Journaling Data Load Instructions	263
	Loading the Connectivity List	265
	For Classic Finite Elements.....	265

	<i>For Face-based Finite Elements</i>	266
	Loading Face Neighbor Data	268
	<i>Local one-to-one</i>	269
	<i>Global or one-to-many</i>	269
	Transient Data	269
	Appending Data with a Data Loader	269
	Replacing Data	270
	Overriding Data Loader Instructions	270
28	<i>Building Extended Curve Fit Add-ons</i>	273
	Registering the External Curve Fit	273
	Calculating the Curve Fit.....	274
	Improving the Probe Value	276
	Providing Curve Fit Information	277
	Curve Fit Settings	277
	Creating the Curve Settings Text Field	278
29	<i>Creating a Data Converter</i>	305
30	<i>Animating</i>	315
	Getting Started	315
	Creating the Dialog.....	316
	Setting Up State Variables/Initializing Dialog Fields	318
	The Animate I Planes Button.....	319
	Writing the AnimatePlanes Function	320
	Monitoring State Changes.....	322
	Exercises	323
A	<i>Building add-ons with FORTRAN</i>	325
	Fortran Include Files	325
	FORTRAN Glue Functions	325
	Language Calling Conventions	326
	<i>Sending String Parameters to Tecplot 360</i>	326
	<i>Receiving String Parameters from Tecplot 360</i>	326
	<i>Handle Parameters</i>	326
	Special Parameter Values.....	327
	Checking FORTRAN Source Using the fcheck Utility.....	327
	Issues on Windows Platforms.....	328
	<i>Calling Conventions</i>	328
	<i>Compiler Directives</i>	328
	<i>Writing to the Console</i>	328

Appendixes

B

<i>Migrating Add-ons</i>	331
Migrating Add-ons.....	331
Active Frame Manipulation	331
Surface Clipping TecUtil Changes	332
Export Format TecUtil Changes.....	332
Load-on-demand for Connectivity Lists and Face Neighbors	332
Migrating Add-ons from Tecplot 360 2006 or Earlier to Tecplot 360 2008 R2 or Later	332
Polyhedral Support.....	333
Shared Grid Files	334
Python Scripting	334
New in Tecplot 360 2013 R1: Functions for Active Frame Manipulation.....	334
New in Tecplot 360 2008: TecUtil Functions.....	335
Migrating Add-ons From Version 10 to Tecplot 360 2006 ...	336
Data Loaders.....	336
Time Aware.....	336
Data Functions	336

Index

.....	339
-------	-----

List of Tutorials

Introduction

Hello World! (Linux/Macintosh Platforms)	26
An Example of an MFC DLL.....	31
An Example of a Non-MFC DLL	33
Create a DLL with FORTRAN	35
Hello World! (UNIX Platforms)	36

General Add-on Components

Adding a Dialog that is Updated When Contour Items Change.....	51
Broadcasting a State Change	55
Broadcasting a State Change	55
Switching the Frame Plot Type to 3D Cartesian.....	57

Data

Accessing Data By Reference (Array)	76
---	----

Accessing Data By Reference (Single Value).....	77
Accessing Data By Reference and Changing Values	77
Equate Add-on	80
Adding Auxiliary Data to a Data Set	108
Setting Style for Multiple Linemaps	112
Setting Style for Multiple Fieldmaps	114
Changing the Line Thickness for a Set of Linemaps	118
Setting the Line Color for all Linemaps.....	118
Creating an XY Line Plot with Lines and Symbols.....	118
Creating an XY Line Plot with Bars.....	119
Creating an XY Line Plot with Multiple Axes	119
Adding a Polynomial Curve Fit.....	121
Activating the Line Legend	122
Creating a Custom Mesh Plot	123
Changing the Mesh Color.....	124
Changing the Contour Type.....	126
Changing the Active Colormap	126
Adding Contour Labels	127
Adding a Contour Legend	128
Creating a Custom Vector Plot.....	130
Creating a Custom Scatter Plot.....	132
Creating a Custom Edge Plot.....	134
Working with Translucency	135

Adding Volume Streamtraces	139
Adding Streamtrace Markers.....	140
Activating Iso-surfaces.....	141
Activating Slices	142
Extracting an Arbitrary Slice	143
Limiting the Points Plotted on a XY Line Plot.....	143
Plotting Nodes on the Surface(s) Only	144
Plotting Boundary Cell Faces Only	145
Plotting Only the 7th J-plane.....	145
Changing the Global Colormap.....	148
Creating a Vector Plot with RGB Coloring.....	149
Including an RGB Legend	150
Changing the Linemap Axis Variable Assignments	151
Changing the Axis Range	152
Adding a Text Label	155
Adding Dynamic Text to your Plot	160
Adding a Square Geometry to your Plot.....	164
Depth Blanking	166
Linking 3D Views between Frames.....	167
Linking Axis Style.....	168
Animating Iso-surfaces	169
Animating Transient Data	171
Rotating the Plot.....	173

Changing the Center of Rotation.....	173
Adjusting the Magnification Factor	174
Changing the Line Thickness.....	176
Using an Offset with the StyleValue Class.....	176
Using Sets with the StyleValue Class.....	176

Lists and Sets

Using the ArgList Class with TecUtilDataSetAddZoneX.....	185
Using Argument Lists.....	186
Use a StringList to Create a Dataset.....	187
Working with StringList_pa as an Output Parameter.....	188
Using String Lists.....	188
Getting the Set of Enabled Zones	192
Looping Through the Members of a Set.....	192
An Alternate Way of Looping through Members of a Set.....	192
Using Sets.....	193

Additional Functionality

Zone Processing	199
Adding Macro Commands to the Equate Add-on.....	201
Data Journaling Code.....	206
Adding Help to the Equate Add-on.....	210
Editing All Objects in the Pick List.....	215
Changing the Color of Text and Geometries in Pick List.....	215

Changing Color of Vectors in Pick List.....	216
---	-----

User Interface

Extending Interactive User Interface Capabilities	239
---	-----

Common Add-ons

Immediate Data Loading without Data Journaling.....	253
Immediate Data Loading with Data Journaling.....	254
Using Auto Load-variable-on-demand	258
Using Load Variable-on-demand	261
Adding Journal Instructions for Data Loading	264
Polynomial Integer Add-on.....	279
Simple Average Add-on.....	290
Simple Spreadsheet Converter.....	306

Appendixes

Part 1 Introduction

Introduction

Tecplot 360 add-ons are executable modules that extend Tecplot 360's basic functionality in a well-defined, systematic way. Add-ons are implemented as compiled function libraries, called shared objects, shared libraries, or dynamic-link libraries (DLLs). Tecplot 360 add-ons can be written in C, C++, or FORTRAN. The source code (including the interface) can be platform independent.

Using the Tecplot 360 Application Programming Interface (API) described in this manual and the function syntax, defined in [Tecplot 360 Add-on Developer's Kit Reference Manual](#), you can create add-ons to generate plots, manipulate or analyze data, or perform a broad variety of specialized tasks. Because the add-ons are shared runtime objects, you do not need to link them into Tecplot 360. As such, you are not limited to using the compilers Tecplot 360 uses, and you do not have to compile (or recompile) large libraries of Tecplot 360 function calls.

The user documentation for Tecplot 360 is divided into these nine books:

- [Add-on Developer's Kit - User's Manual](#) (this document) - This manual provides instructions and examples for creating add-ons for Tecplot 360.
- [Getting Started Manual](#) - New Tecplot 360 users are encouraged to work through the tutorials provided in the Getting Started Manual. These tutorials highlight how to work with key features in Tecplot 360.
- [User's Manual](#) - This manual provides a complete description of working with Tecplot 360 features.
- [Scripting Guide](#) - This guide provides Macro and Python command syntax and information on working with Macro and Python files and commands.
- [Quick Reference Guide](#) - This guide provides syntax for zone header files, macro variables, keyboard shortcuts, and more.
- [Data Format Guide](#) - This guide provides information on outputting simulator data to Tecplot 360 file format.
- [Add-on Developer's Kit - Reference Manual](#) - This manual provides the syntax for the functions included in the add-on kit.
- [Installation Instructions](#) - These instructions give a detailed description of how to install Tecplot 360 on your machine.

- [Release Notes](#) - These notes provide information about new and/or updated Tecplot 360 features.
- **Tecplot Talk** - A user-supported forum discussing Tecplot 360, Tecplot Focus, Python scripting, Add-on development, TecIO and more. Visit www.tecplottalk.com for details.

In addition to the user documentation and a set of include files and libraries, the Add-on Developer's Kit (ADK) includes the following components:

- **Tecplot 360 GUI Builder** - a platform independent tool for creating the user interface for your add-on. Refer to [Chapter 25: "Tecplot GUI Builder"](#) for additional information.
- **Samples directory** - a collection of sample add-ons. These add-ons are included in your Tecplot 360 distribution in `$TEC_360_2013R1/adk/samples`. A separate collection of sample add-ons for Visual Studio 2005 are included in `$TEC_360_2013R1/adk/vs2005samples`.
- **Add-ons available on Tecplot Talk** - Refer to <http://www.tecplottalk.com/addons.php> for additional add-ons available for download. These add-ons are developed by Tecplot 360 customers, as well as Tecplot 360 developers. Please note that these add-ons are not subject to our Quality Assurance processes and are not covered by any Technical Support agreement. Please use the forums available on Tecplot Talk for questions and comments.
- **Tecplot Toolbox** - The Tecplot Toolbox provides an alternate method for communicating with the Tecplot Engine. It is a convenience library that provides an object-oriented wrapper for many of the procedural commands housed in the Tecplot Engine, including State Change commands and style commands. To use the Tecplot Toolbox, you must include `tptoolbox.h` in your project. We recommend working with the Toolbox code in lieu of the Classic Code. The `StyleValue` and `AuxData` classes are simpler to use and also require a fraction of the code.

The following classes are included in the Tecplot Toolbox (and are described in their subsequent sections):

- **Argument Lists** - [Chapter 16: "Argument Lists"](#)
- **Auxiliary Data** - [Chapter 12: "Auxiliary Data"](#)
- **State Changes** - [Chapter 6: "State Changes From an Add-on"](#)
- **Sets** - [Chapter 18: "Sets"](#)
- **String Lists** - [Chapter 17: "String Lists"](#)
- **Style Values** - [Chapter 13: "Setting Plot Style"](#)

1 - 1 Platform Issues

Different operating systems have different ways of creating and using shared libraries. Tecplot 360's Add-on Developer's Kit (ADK) provides utilities that mask most of these differences for related programs. All Windows, Linux or Macintosh systems will behave in a similar fashion. ADK tools will resolve the differences for you. All of the examples of the source code shown in this manual are included in the Tecplot 360 distribution and are found in the `adk/samples` subdirectory below the Tecplot 360 home directory.

1 - 2 Add-ons Created with Previous Versions

If you are working with add-ons created with previous versions of Tecplot 360, Tecplot Focus or Tecplot and you wish to use the add-on with a Tecplot 360 version 2013 or Tecplot Focus version 2013 on a Windows machine, you must update the add-on source code to link with `libtec.lib` (in lieu of `tecplot.lib`).

1 - 3 Recent Updates

The following `TecUtil` functions have been deprecated as of Tecplot 360 2008 Release 2:

`TecUtilDataNodeGetRawPtr`

TecUtilDataNodeGetRef
 TecUtilDataFaceNbrArrayAssign
 TecUtilDataFaceNbrAssign
 TecUtilDataFaceNbrBeginAssign
 TecUtilDataFaceNbrBeginAssignX
 TecUtilDataFaceNbrEndAssign
 TecUtilDataFaceNbrGetByRef
 TecUtilDataFaceNbrGetByZone
 TecUtilDataFaceNbrGetRawPtr
 TecUtilDataFaceNbrGetRef

When upgrading from Tecplot 360 2008 Release 1 or earlier to Tecplot 360 2008 Release 2 or later, the following items also require consideration:

- InitTecAddOn has been deprecated and replaced with InitTecAddOn113.
- C++ users - InitTecAddOn will automatically convert to InitTecAddOn113 upon recompiling.
- Fortran users - You must manually change instances of InitTecAddOn to InitTecAddOn113 before recompiling.
- If you update Tecplot 360 on a Windows machine, you must update your non-Tecplot add-ons to link with *libtec.lib* (instead of *tecplot.lib*).

1 - 4 Macros Versus Add-ons

In addition to the add-on libraries, Tecplot 360 provides an extensive set of macro commands. Macros should be used for simple repetitive tasks that can automate Tecplot 360, while add-ons should be written for complex tasks that augment Tecplot 360.

You should write an add-on if your task requires any of the following functions:

- Persistence of the state of Tecplot 360 (i.e. recalling the last file read, state of toggles, etc.)
- Extensive calculations
- Communication/interaction with other applications
- Loading custom data files
- Custom curve fits
- A complex user interface, such as: single selection lists, toggles, sidebars or modal dialogs
- Operation on a current set of picked items in Tecplot 360

You should use a set of macro commands, for simple, automated tasks that do not involve picked objects. Refer to the [Scripting Guide](#) for complete details on writing macro commands for Tecplot 360.

Creating Add-ons on Linux/ Macintosh Platforms

The Tecplot 360 Add-On Developers Kit (ADK) contains include files (.h) along with a *libtec.lib* file with which to link add-on source code. To create an add-on for Tecplot 360 with the Add-on Developer's kit, you must perform the following steps:

- First, follow the steps in [Section 2 - 1 "General Setup for Creating Add-ons"](#) for setting up your software and environment variables. The steps described in this section only need to be performed once.
- Second, follow the steps in [Section 2 - 2 "Creating a New Add-on"](#) for setting up your development project. The steps performed in this section are required for each add-on that you write.
- Third, develop your add-on using the TecUtil function provided in the ADK. The syntax for these functions is provided in the [ADK Reference Manual](#).
- Fourth, compile your add-on using the instructions provided in [Section 2 - 3 "Compiling the Add-on"](#).

2 - 1 General Setup for Creating Add-ons

To create add-ons in Tecplot 360 you must set up a working directory where source code can be created and edited. This directory will be referred to as the *Add-on Development Root Directory*.

In preparation for writing add-ons, perform the following steps:

1. Install Tecplot 360.
2. Create the Add-on Development Root Directory.
3. Verify that the \$TEC_360_2013R1 environment variable is assigned to the directory where Tecplot 360 was installed.
4. Verify that the PATH environment variable includes the following:
\$TEC_360_2013R1/bin:\$TEC_360_2013R1/adk/bin.
5. Create a new file called *tecdev.add* in the *Add-on Development Root Directory* and add the following line to the file:
#!/MC 1120

6. [Optional] If you plan to use the [Tecplot GUI Builder](#), add the following line to the *tecdev.add* file in your Add-on Development Root Directory:

```
$!LoadAddon "|TECHOME|/lib/libguibld"
```



This step is recommended for multi-platform development.

7. Set the environment variable *TECADDONDEVDIR* to the path of the *Add-on Development Root Directory*.
8. Set the environment variable *TECADDONDEVPLATFORM* to one of the valid platform names. A list of valid platforms can be obtained by running `tec360 -platlist`.
9. Set the environment variable *TECADDONFILE* to the path of your add-on file(s) (e.g. *tecplot.add* or *tecdev.add*).

Once the preceding steps are completed, you are ready to proceed with [Section 2 - 2 "Creating a New Add-on"](#).

2 - 2 Creating a New Add-on

The setup performed in [Section 2 - 1 "General Setup for Creating Add-ons"](#), allows your installation of Tecplot 360 to communicate with any add-ons that you may write. The steps only need to be performed only once.

This section discusses the preliminary steps taken to begin writing an add-on. These steps should be performed for every add-on that you write.

To create a new add-on, perform the following steps:

1. Go to the *Add-on Development Root Directory*.
2. Type: `CreateNewAddOn`

This will ask you a few questions about the add-on to be built, including whether or not you intend to use the Tecplot GUI Builder. When this is finished, you will have a new subdirectory named *MyAddOnName*, where *MyAddOnName* is the name that you supplied while running `CreateNewAddOn`. This subdirectory contains a set of files that can be compiled to create a minimal add-on.

3. Edit the *tecdev.add* file located in the *Add-on Development Root Directory* and add the following line:

```
$!LoadAddon "|$TECADDONDEVDIR|/libMyAddOnName"
```

4. Edit the set of files created in *Add-on Development Root Directory/MyAddOnName* to add functionality to your add-on.

In order for your add-on to communicate with Tecplot 360, it must do the following:

1. **Initialize the Add-on** - Make the "initialization" function named `InitTecAddOn` public. The initialization function typically includes a call to add a converter, add a loader, register a curve fit, or add an item to the **Tools** menu. When you run `CreateNewAddOn`, this function is automatically added to the file *main.c* (or *main.cpp*). When Tecplot 360 launches, it scans the *tecdev.add* file, loads named shared object libraries and calls the `InitTecAddOn` function. Refer to [Chapter 5: "Add-on Initialization and Cleanup"](#) for additional information.
2. **Make calls to the TecUtil functions** - The `TecUtil` functions allow you to do a wider range of tasks than those available through the Tecplot 360 interface itself. They are available from the `libtec` shared object library. Refer to [ADK Reference Manual](#) for details on the `TecUtil` functions.

3. **[Optional] Write the GUI Code** - The Tecplot 360 Add-on Developers Kit includes a simple GUI builder called Tecplot GUI Builder (TGB). When you run `CreateNewAddOn` and choose to use the TGB, a starter set of TGB files is created for you. Refer to [Chapter 25: "Tecplot GUI Builder"](#) for additional information.

You are not restricted to this GUI builder. You may use a commercial GUI builder such as *Builder Xcessory* or *X-Designer*. When you run `CreateNewAddOn` and choose to use the TGB, a starter set of TGB files is created for you.

From this point on, when you want to test the add-ons you are developing, use the `-develop` flag when running Tecplot 360. When you have completed coding your add-on, copy the shared object library to the `lib` subdirectory below the Tecplot 360 home directory and include the command:

```
$!LoadAddOn "|$TEC_360_2013R1|/lib/libMyAddOnName"
```

in the `tecplot.add` file in the Tecplot 360 home directory.

Refer to ["Hello World! \(Linux/Macintosh Platforms\)"](#) on page 26 for an introductory tutorial on creating a Tecplot 360 Add-on.

2 - 3 Compiling the Add-on

2 - 3.1 Using Runmake

If you used `CreateNewAddOn`, compiling the add-on is straightforward. Go to the subdirectory where your add-on source code is located and type:

```
Runmake
```

You can choose the platform type and what type of executable to create. For example, to compile on an SGI machine under IRIX 6.5 and create a debug version use:

```
Runmake sgix.65 -debug
```

To make a release version use:

```
Runmake sgix.65 -release
```

If your compile is successful, you will end up with a shared object library located in `../lib/platform/buildtype`. Running Tecplot 360 with the `-develop` flag automatically directs it to look for your library in this directory.



If the Tecplot 360 home directory and your Add-on Development Directory are located in directories that can be remotely mounted by other UNIX® or Linux® computers, then you can log on to those computers and use `Runmake` as described earlier. The resulting shared library will be stored in the appropriate subdirectory for the computer platform.

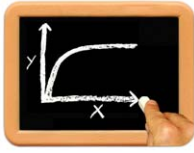
2 - 3.2 Editing the CustomMake File

The `Runmake` command used to build your add-on actually invokes the Linux `make` program with a large list of flags that customize the make process for your platform. Prior to calling `make`, the `Runmake` shell script checks to see if a local file called `CustomMake` exists and is executable. If so, it runs the `CustomMake` shell script in place and then runs `make`. This process allows you to add to or completely replace any assignments made by `Runmake`.

For example, to add an additional flag called `-xg` to the `cc` compile command, edit the local `CustomMake` shell script (in the subdirectory of your add-on) by adding:

```
CFLAGS="$CFLAGS -xg"
```

This replaces `CFLAGS` (i.e. the flags used with the `cc` command) with its original contents plus the `-xg` flag. The default `CustomMake` file created in your add-on directory when you run `CreateNewAddOn` contains edit instructions including an explanation of the flags available to change.



Example 1: Hello World! (Linux/Macintosh Platforms)

This *Hello World* example uses source code files created by the `CreateNewAddOn` script (UNIX). To complete this example on a Windows platform, use the example files found in `$TEC_360_2013R1\adk\vs2005samples`. Our project name will be “hiwrld”, and the add-on name will be “Hello World.”

Step 1 Add-on Setup

When running `CreateNewAddOn`, answer the questions as follows:

- **Project name (base name)** - hiwrld
- **Add-on name** - Hello World
- **Company name** - [Your Company Name]
- **Type of Add-on** - General Purpose
- **Language** - C
- **Use TGB to create a platform-independent GUI?** No
- **Add a menu callback to the Tecplot "Tools" menu?** Yes
- **Menu Text** - Hello World!

The question “Use TGB to create a platform-independent GUI” option specifies that you will use Tecplot GUI Builder in your add-on. After running the `CreateNewAddOn` script, you should have the following files: `ADDGLBL.h` and `main.c`. You will have other files specific to your platform, but only those above will be modified. Verify that you can compile your add-on project and load it into Tecplot 360, by running the `Runmake` script. Once you have compiled *Hello World*, run Tecplot 360 and select “Hello World!” from Tecplot’s **Tools** menu. Text is written to standard out, reading *Menu function called*. When finished, this will read “Hello World!” in a dialog.

Step 2 Modifying the MenuCallback() function

Most add-ons contain a callback function named `MenuCallback()`. Tecplot 360 calls this when the user selects an add-on from the **Tools** menu in the Tecplot interface. This callback function is registered by the `TecUtilMenuAddOption()` function, which is in `InitTecAddOn()`. Modify the source code, as shown below, to display “Hello World” in the dialog. Refer to the [ADK Reference Manual](#) for the syntax of the `TecUtil` functions used in this example.

In `main.c`, edit the `MenuCallback()` function as follows:

```
static void STDCALL MenuCallback(void)
{
    TecUtilLockStart(AddOnID);
    TecUtilDialogMessageBox("Hello World!", MessageBox_Information);
    TecUtilLockFinish(AddOnID);
}
```

You have now completed *Hello World*. Recompile and run Tecplot 360.



2 - 4 Debugging Shared Objects

When debugging shared objects, we recommend you follow this procedure:

1. Compile a debug version of the add-on.
2. Start up Tecplot 360 with the debug enabled add-on loaded.
3. In another shell, go to the source directory where you have your add-on and type the following:

```
ddd $TEC_360_2013R1/bin/tecplot.shared
```
4. Choose "File/Attach to Process". ddd should automatically "pick" the correct process to attach to. Click the Attach button.
5. At this point you are debugging the Tecplot 360 application which has your add-on attached. Any gdb command will work.
6. One useful thing to do is

```
(gdb) list mymodule.c:100
```

This will dive into the module called "mymodule.c" starting at line 100. In ddd you can then scroll through the source and set breakpoints etc. When you are ready to scroll, click the Cont button.

Creating Add-ons on Windows Platforms

The Tecplot 360 Add-On Developers Kit (ADK) contains include files (.h) along with a *libtec.lib* file with which to link add-on source code. The *libtec.lib* library will dynamically load *libtec.dll* at runtime. To create an add-on for Tecplot 360 with the Add-on Developer's kit, you must perform the following steps:

1. First, follow the steps in [Section 3 - 1 "General Steps for Creating an Add-on"](#) for setting up your software and environment variables. You need to perform the steps described in this section only once.
2. Follow the steps in [Section 3 - 2 "Creating an Add-on with Visual Studio 2005"](#) or [Section 3 - 3 "Creating an Add-on with Intel Visual Fortran"](#) to set up your development project. Each add-on that you write requires the steps in one of these sections.
3. Develop your add-on using the TecUtil function provided in the ADK. The syntax for these functions is provided in the [ADK Reference Manual](#).



The Tecplot 360 ADK is supplied with dynamic-link libraries created by Microsoft®. This is in compliance with the Visual Studio® license agreement. The license agreement, however, also states that licenses cannot be transferred a second time unless the party distributing the libraries also has a Visual Studio license agreement. In other words, if you develop a Tecplot 360 add-on and you plan to distribute it outside of your organization, you must also have the right to distribute the Microsoft dynamic link libraries. If you own Microsoft Visual Studio, you have this right to distribution.

3 - 1 General Steps for Creating an Add-on

To setup your system for building add-ons, perform the following steps:

1. Install Tecplot 360. If you choose the custom installation, include the Add-on Developers Kit option. The SETUP program will automatically set your *\$TEC_360_2013R1* environment variable and include the *bin* subdirectory, below the Tecplot 360 home directory, in your path.
2. If you plan on using Tecplot GUI Builder (TGB), make sure the following line is in the *tecplot.add* file in the Tecplot 360 home directory:

```
$!LoadAddon "guibld"
```

Tecplot GUI Builder is discussed in detail in [Chapter 25: "Tecplot GUI Builder"](#).

3 - 2 Creating an Add-on with Visual Studio 2005

You should be familiar with Microsoft® Visual Studio® 2005, and its concepts, such as DLLs and callback functions, before reading this section.



To complete any example in the Add-on Developer's Kit User's Manual that uses files generated by the CreateNewAddOn script, use the example files found in `$TEC_360_2013R1\adk\vs2005samples`.

3 - 2.1 Creating an Add-on Using Visual Studio

Follow these guidelines to create an add-on with Visual Studio. The next section gives guidelines for debugging your add-on.

1. To create an MFC add-on for Tecplot 360 using Visual Studio, your project workspace must be of type "MFC Application" or "MFC DLL". You can choose this when starting a new project workspace (**File>New>Project**). You must choose "Regular DLL using shared MFC DLL" when prompted for the type of DLL.
2. In order to have access to Tecplot 360's functions and data, your project must find the Tecplot 360 include files. These files are located in the *include* subdirectory of the Tecplot 360 home directory. To include this subdirectory in your project:
 - a. Open the **Properties** dialog from the **Project** menu.
 - b. In the left-hand navigation pane, go to **Configuration Properties>C/C++>General**.
 - c. On the General page, add `$TEC_360_2013R1\include` as an "Additional Include Directories", replacing `$TEC_360_2013R1` with the location of your Tecplot 360 installation directory¹. This will include Tecplot 360's *include* subdirectory within the Tecplot 360 home directory.

These files declare Tecplot 360's functions and data types so that they will be available to your add-on. The functions (with names starting with TecUtil) allow you to do any tasks Tecplot 360 can do, and other tasks that meet your specifications.

3. Next, make *libtec.lib* available to your project to resolve Tecplot 360 functions when your project links to Tecplot 360. To do this, open the **Properties** dialog from the **Project** menu, and follow one of these two options to input the information:
 - a. In the navigation pane, go to **Linker>Input**. On the Input page, click the Browse button next to Additional Dependencies, and add `[path to libtec.lib]\libtec.lib` as a dependency.
 - b. Alternatively, go to **Linker>Input**, and add just *libtec.lib* as one of the "Additional Dependencies" on the Input page. Then, navigate to **Linker>General**, and on the General page, add `[path to libtec.lib]` under "Additional Library Directories".
4. In order for your add-on to communicate with Tecplot 360, it must export a STDCALL initialization function called `InitTecAddOn113`. In Visual Studio 2005, the function should look as follows:

```
EXPORTFROMADDON void STDCALL InitTecAddOn113 (void)
{
    .
    .
    .
}
```

5. If you are using MFC, add the following line your initialization function and any other functions which Tecplot 360 will call:

1. For Windows users, `$TEC_360_2013R1` is typically `C:\Program Files\Tecplot\Tec360 2013R1`.

MANAGESTATE

6. To test your add-on with Tecplot 360, create your DLL file by building your project (choose Build Solution from the **Build** menu). Make sure you know the location of your DLL file.
7. To test your add-on with Tecplot 360, run Tecplot 360 from your DLL project. To do this:
 - a. Open the **Properties** dialog from the **Project** menu.
 - b. In the navigation pane, choose “Debugging”, underneath the “Configuration Properties”.
 - c. Set the “Command” to `$TEC_360_2013R1\tec360.exe`.
 - d. Set the “Command Arguments” to `-loadaddon [path to project_name.dll]`.
 - e. Set the “Working Directory” to “Debug”.

You can now set breakpoints to debug your add-on.

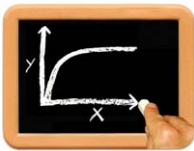


We have provided skeleton code of the basic types of add-ons (General Purpose, Data Loader, Data Converter and Curve Fit) for add-on developers using Visual Studio 2005, in `$TEC_360_2013R1\adk\vs2005samples`. Each sample project contains a *readme.txt* describing the project settings and code included in the project.

3 - 2.2 Debugging Your Add-on

If your add-on does not compile, use the following methods to aid in debugging:

1. Error messages displayed by Tecplot 360 while you are running your add-on may aid in debugging.
2. Tecplot 360 expects that all of the functions that are passed to it from an add-on are STDCALL. STDCALL is automatically included if you use the EXPORTFROMADDON keyword. However, this is not the default calling convention in Visual Studio.
3. Use the DUMPBIN utility to see what functions your DLL exports. You can do this with the DUMPBIN utility with the /EXPORTS flag. (For example, “DUMPBIN /EXPORTS myaddon.dll”.) If InitTecAddOn113 is not listed, Tecplot 360 will not be able to access it.
4. If you are using an MFC DLL, make sure you are using MFC in a shared library. Also check your preprocessor definitions: _AFXDLL must be defined and _USRDLL must not be defined.



Example 2: An Example of an MFC DLL

Follow this example to create an MFC DLL:

1. To begin creating an MFC add-on for Tecplot 360 using Visual Studio, start with a new project workspace, (using **File>New>Project**). Choose “MFC DLL” as the project type. Name the project “SimpMFC”. When prompted, select “Regular DLL using shared MFC DLL”. (This example assumes `C:\projects\simpmfc` represents the location of your project and `$TEC_360_2013R1` represents your Tecplot 360 installation directory. Substitute the names of your own project directories in the following example.)
2. To add a dialog to your add-on, select “Add Resource” from the **Project** menu. In the **Add Resource** dialog, select “Dialog” and select [New] to insert the new dialog.

3. Select "Static Text" in the **Dialog Editor** and add text to read "This is an MFC add-on." (If the **Dialog Editor** is not open, go to **View>Toolbox** to display it.)
4. Using the dialog's **Properties** toolbox, change the dialog's ID from "IDD_DIALOG1" to "IDD_ADDONDLG" and the Caption to "Simple MFC Add-On".
5. In order to use the dialog in the add-on, create a class for it. Double-click on the dialog to bring up the Class Wizard. Type in the Class name "CSimpDlg." Select a Base Class of "CDialog". (The Dialog ID "IDD_ADDONDLG" will fill in automatically.) Select [Finish] to create the class and close the Class Wizard.
6. Edit the file *SimpMFC.cpp*: near the top of the file, immediately after the line "#include SimpMFC.h", add the following lines:

```
#include "TECADDON.h"
#include "SimpDlg.h"
Addon_pa COMMANDPROCESSORID;
```

Near the bottom of the file, after the line "CSimpMFCApp theApp", add the following lines:

```
static void STDCALL LaunchSimpleDialog(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState( ));
    CSimpDlg modal;
    modal.DoModal;
}
EXPORTFROMADDON void STDCALL InitTecAddOn113 (void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState( ));
    TecUtilLockOn();
    COMMANDPROCESSORID=TecUtilAddOnRegister(110,"Simple MFC Test",
                                           "1.0",
                                           "My Company");
    TecUtilMenuAddOption("Tools",
                        "Simple MFC Addon",
                        'S',
                        LaunchSimpleDialog);
    TecUtilLockOff();
}
```

7. In the **Properties** dialog (open from the **Project** menu), make the following changes:
 - a. In the navigation pane, select "Debugging", underneath the "Configuration Properties".
 - b. Set the "Command" to `$TEC_360_2013R1\bin\tec360.exe`.
 - c. Set the "Command Arguments" to `"-loadaddon [path to simpmfc.dll]"`.
 - d. Set the "Working Directory" to "Debug".
 - e. In the navigation pane, go to **Configuration Properties>C/C++>General**.
 - f. On the General page, Add `$TEC_360_2013R1\include` as one of the "Additional Include Directories".
 - g. Go to "Input" under "Linker" in the navigation pane. On the Input page, add `$TEC_360_2013R1\bin\libtec.lib` as one of the "Additional Dependencies".
8. Build the debug version of your project and run it (by selecting "Build Solution" from the **Build** menu). Launch Tecplot 360, go to the **Tools** menu, and select "Simple MFC Add-on". This action will launch your dialog.



Example 3: An Example of a Non-MFC DLL

Following this example to create a non-MFC DLL:

1. To begin creating an add-on for Tecplot 360 using Visual Studio, open a new project (**File>New>Project**). Choose "Visual C++" in the left navigation pane, select "Win32" under the "Visual C++" category, and select the project type "Win32 Project". Name the project "Simple". This example assumes `C:\projects\simple` represents the location of your project and `$TEC_360_2013R1` represents your Tecplot 360 installation directory. Substitute the names of your own project directories in the example following. In the Win32 Application Wizard, select "DLL" as the Application Type.
2. Open a new C/C++ file (**File>New>File>Visual C++>C++ File**), and type in the following lines. Then, save the file as "*SIMPLE.C*".

```
#include "TECADDON.h"
AddOn_pa COMMANDPROCESSORID;
static void STDCALL LaunchSimpleDialog(void)
{
    TecUtilLockStart(COMMANDPROCESSORID);
    TecUtilDialogMessageBox("This is the Simple dialog!",
        MessageBox_Information);
    TecUtilLockFinish(COMMANDPROCESSORID);
}
EXPORTFROMADDON void STDCALL InitTecAddOn113(void)
{
    TecUtilLockOn();
    COMMANDPROCESSORID=TecUtilAddOnRegister(110, "Simplenon-MFCTest",
        "1.0", "My Company");
    TecUtilMenuAddOption("Tools",
        "Simple non-MFC Test",
        'S',
        LaunchSimpleDialog);
    TecUtilLockOff();
}
```

3. In the **Properties** dialog, on the "Debugging" page under "Configuration Properties", set "Command" to `$TEC_360_2013R1\bin\tec360.exe`. Set "Command Arguments" to `-loadaddon [path to simple.dll]`. Set the "Working Directory" to "Debug".
4. On the General page, Add `$TEC_360_2013R1\include` as one of the "Additional Include Directories".
5. Go to "Input" under "Linker" in the navigation pane. On the Input page, add `$TEC_360_2013R1\bin\libtec.lib` as one of the "Additional Dependencies".
6. Add the *SIMPLE.C* file to your project.

Build the debug version of your project and then run it. When Tecplot 360 comes up, go to the **Tools** menu and select "Simple non-MFC Test." Your dialog will launch with the message, "This is the Simple dialog!"

3 - 3 Creating an Add-on with Intel Visual Fortran



Tecplot 360 supports FORTRAN on Windows operating systems only if you are using Intel Visual FORTRAN.

This section is for users using Intel Visual FORTRAN 6.0 or later. You should be familiar with its use and concepts (such as DLLs) before using the information provided in this section.

If you are not familiar with this process, please refer to your Visual FORTRAN documentation and online help. We recommend you go through several examples of creating DLLs before attempting to create an Tecplot 360 add-on.

3 - 3.1 Creating an Add-on Using Visual Fortran

1. To create an add-on for Tecplot 360 using Visual FORTRAN, use Application Type “DLL”. Select this in the new project wizard, after choosing a Project template.
2. In order to have access to Tecplot 360's functions and data, your project must find the Tecplot 360 include files. These files are located in the *include* subdirectory of the Tecplot 360 home directory. To include this subdirectory in your project:
 - a. Open the **Properties** dialog from the **Project** menu.
 - b. In the left-hand navigation pane, go to **Configuration Properties>C/C++>General**.
 - c. On the General page, add *\$TEC_360_2013R1\include* as an “Additional Include Directories”. This will include Tecplot 360's *include* subdirectory, within the Tecplot 360 home directory.

These files declare Tecplot 360's functions and data types so that they will be available to your add-on. The functions (with names starting with TecUtil) allow you to do any tasks Tecplot 360 can do, and other tasks that meet your specifications.

3. Next, make *libtec.lib* and *fglue.lib* available to your project to resolve Tecplot 360 functions when your project links to Tecplot 360. To do this, open the **Properties** dialog from the **Project** menu. In the navigation pane, go to **Linker>Input**. On the Input page, add the two libraries as “Additional Dependencies”. As an alternate method, navigate to **Linker>General**, and on the General page, add the path to each of the two libraries under “Additional Library Directories”. The libraries are located in the *bin* subdirectory of the Tecplot 360 home directory.
4. In order for your add-on to communicate with Tecplot 360, it must export an initialization subroutine called *InitTecAddOn113* (case is not important). This subroutine should look as follows:

```
subroutine InitTecAddOn113
!DEC$attributes DLLEXPORT::InitTecAddOn113
.
.
.
end
```

5. When you are ready to test your add-on with Tecplot 360, create your DLL file by building your project (“Solution”). Make sure you know the location of your DLL file. When you build the Debug version of your project, you may get a warning message recommending you use the “/NODEFAULTLIB” link option. You may ignore this warning. If you wish to eliminate it, check the “Ignore all default libraries” link option, and explicitly add the additional FORTRAN and C libraries required (refer to your FORTRAN documentation for a list of these).

To run your add-on with Tecplot 360, run Tecplot 360 from your DLL project. To do this:

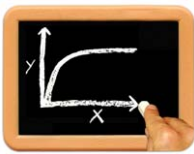
1. In the navigation pane of the **Properties** dialog, select "Debugging", underneath the "Configuration Properties".
2. Set the "Command" to `$TEC_360_2013R1\bin\tec360.exe`.
3. Set the "Command Arguments" to `-loadaddon [path to project_name.dll]`.
4. Set the "Working Directory" to "Debug".
5. In the navigation pane, go to **Configuration Properties>C/C++>General**.
6. On the General page, Add `$TEC_360_2013R1\include` as one of the "Additional Include Directories".
7. Go to "Input" under "Linker" in the navigation pane. On the Input page, add `$TEC_360_2013R1\bin\libtec.lib` as one of the "Additional Dependencies".

You can now set a breakpoint anywhere in your code to debug your add-on.

3 - 3.2 Debugging Your Add-on

If your add-on does not compile, use the following methods to aid in debugging:

8. Error messages displayed by Tecplot 360 while your add-on runs may help.
9. Tecplot 360 expects that all of the functions and subroutines passed to it from an add-on are STDCALL. This is the default calling convention in Visual FORTRAN version 6 (you should *not* explicitly set the STDCALL attribute for these subroutines).
10. Use the DUMPBIN utility to view the functions being exported by your DLL. You can do this with the DUMPBIN utility with the /EXPORTS flag. (For example, "DUMPBIN/EXPORTS myaddon.dll".) If INITTECADDON113 is not listed, Tecplot 360 cannot access it. The name of your initialization function will be decorated as follows: "_INITTECADDON113@0". If it is not, make sure you are compiling using the default External Procedures options for FORTRAN. To do this, go to the External Procedures page under "Fortran" in the navigation pane of the **Properties** dialog, and set the "Calling Convention" and "Name Case Interpretation" to "Default".
11. Make sure you have set the DLLEXPORT attribute as shown in [Section 3 - 3.1 "Creating an Add-on Using Visual Fortran"](#).



Example 4: Create a DLL with FORTRAN

Following this example to create a DLL with FORTRAN:

1. To begin creating an add-on for Tecplot 360 using Visual FORTRAN, start with a new project workspace. Select the project type "Intel(R) Fortran Projects", and select the "Dynamic-link Library" template. Name the project "Simpfor". This example assumes that `C:\projects\simpfor` represents the location of your project and that `$TEC_360_2013R1` represents your Tecplot 360 installation directory. Substitute the names of your own project directories in the following example.
2. Add a new FORTRAN source file to your project, name it `"SIMPFOR.F"` and add the following lines:

```
subroutine LaunchSimpforDialog
include "FGLUE.INC"
common /Addon/ COMMANDPROCESSORID
```

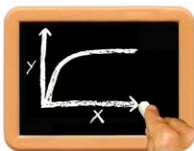
```

pointer (COMMANDPROCESSORID, AddonUnused)
integer i
call TecUtilLockStart(COMMANDPROCESSORID)
i = TecUtilDialogMessageBox(
& "This is the Simpfor dialog!"//char(0),
&  MessageBox_Information)
call TecUtilLockFinish(COMMANDPROCESSORID)
return
end

subroutine InitTecAddOn113
!DEC$ attributes DLLEXPORT::InitTecAddOn113
include "FGLUE.INC"
common /Addon/ COMMANDPROCESSORID
pointer (COMMANDPROCESSORID, AddonUnused)
integer i
external LaunchSimpforDialog
call TecUtilLockOn
call TecUtilAddOnRegister(110,
& "Simple FORTRAN Test"//char(0),
& "1.0"//char(0),
& "My Company"//char(0),
& COMMANDPROCESSORID)
i = TecUtilMenuAddOption(
& "Tools"//char(0),
& "Simple FORTRAN Test"//char(0),
& 'S'//char(0),
& LaunchSimpforDialog)
call TecUtilLockOff
return
end

```

3. On the Debugging page underneath "Configuration Properties" in the **Properties** dialog:
 - Set the "Command" to \$TEC_360_2013R1\bin\tec360.exe.
 - Set the "Command Arguments" to "-loadaddon [path to *project_name.dll*]".
 - Set the "Working Directory" to "Debug".
4. Go to "Input" under "Linker" in the navigation pane. On the Input page, add \$TEC_360_2013R1\bin\libtec.lib and \$TEC_360_2013R1\bin\fglue.lib as "Additional Dependencies".
5. On the Preprocessor tab under "Fortran", add \$TEC_360_2013R1\include to the "Additional Include Directories". (Separate this from any other listed paths by a semi-colon).
6. Build the debug version of your project and run it. When Tecplot 360 launches, go to the **Tools** menu and select "Simple FORTRAN Test." Your dialog will launch with the message, "This is the Simpfor dialog!"



Example 5: Hello World! (UNIX Platforms)

This *Hello World* example uses source code files created by the CreateNewAddOn script (UNIX). To complete this example on a Windows platform, use the example files found in \$TEC_360_2013R1\adk\vs2005samples. Our project name will be "hiwrl", and the add-on name will be "Hello World."

Step 1 Add-on Setup

When running CreateNewAddOn, answer the questions as follows:

- **Project name (base name)** - hiwrlld
- **Add-on name** - Hello World
- **Company name** - [Your Company Name]
- **Type of Add-on** - General Purpose
- **Language** - C
- **Use TGB to create a platform-independent GUI?** No
- **Add a menu callback to the Tecplot "Tools" menu?** Yes
- **Menu Text** - Hello World!

The question "Use TGB to create a platform-independent GUI" option specifies that you will use Tecplot GUI Builder in your add-on. After running the CreateNewAddOn script, you should have the following files: *ADDGLBL.h* and *main.c*. You will have other files specific to your platform, but only those above will be modified. Verify that you can compile your add-on project and load it into Tecplot 360, by running the Runmake script. Once you have compiled *Hello World*, run Tecplot 360 and select "Hello World!" from Tecplot's **Tools** menu. Text is written to standard out, reading *Menu function called*. When finished, this will read "Hello World!" in a dialog.

Step 2 Modifying the MenuCallback() function

Most add-ons contain a callback function named `MenuCallback()`. Tecplot 360 calls this when the user selects an add-on from the **Tools** menu in the Tecplot interface. This callback function is registered by the `TecUtilMenuAddOption()` function, which is in `InitTecAddOn()`. Modify the source code, as shown below, to display "Hello World" in the dialog. Refer to the [ADK Reference Manual](#) for the syntax of the `TecUtil` functions used in this example.

In *main.c*, edit the `MenuCallback()` function as follows:

```
static void STDCALL MenuCallback(void)
{
    TecUtilLockStart(AddOnID);
    TecUtilDialogMessageBox("Hello World!", MessageBox_Information);
    TecUtilLockFinish(AddOnID);
}
```

You have now completed *Hello World*. Recompile and run Tecplot 360.



Running Tecplot 360 with Add-ons

When Tecplot 360 is started, it goes through various initialization phases, including the processing of the *tecplot.cfg* file, the loading of the Tecplot 360 stroke font file (*tecplot.fnt*), and the initialization of the graphics. When these steps have been completed, Tecplot 360 looks for add-ons.

4 - 1 Specifying which Add-ons to Load

Using different commands when you launch Tecplot 360, you can customize the list of add-ons to be loaded by different Tecplot 360 users on your network, or by a single user.

4 - 1.1 Add-ons Loaded by All Users

In a typical installation of Tecplot 360, the add-ons you want to be loaded by all users of Tecplot 360 are named in an add-on load file called *tecplot.add*, located in the Tecplot 360 home directory¹. The *tecplot.add* file started with the macro header line (`#!MC 1120`) and contains only `#!LoadAddOn` commands. The following is an example of a typical *tecplot.add* file:

```
#!MC 1120
#!LoadAddOn "myaddon"
```

4 - 1.2 Specifying a Secondary Add-on Load File

You may also instruct Tecplot 360 to load a different list of add-ons by naming a second add-on load file using one of the following methods:

- Include `-addonfile addonfilename` on the command line, or
- Set the environment variable `TECADDONFILE` to point to your add-on directory.

4 - 1.3 Specifying Add-ons Via the Command Line

You can also instruct Tecplot 360 to load a particular add-on via the command line. By including the file name (include path and extension) of the add-on in the command line, you may specify as many add-ons as you want. After add-ons are loaded, Tecplot 360 re-processes all command line arguments not

¹. For Windows users, this is typically: `C:\Program Files\Tecplot\Tec360 2013R1`.

processed earlier (for graphics and add-on initialization). This ordering allows for a data reader add-on (discussed later) to be used to load data specified on the command line.

4 - 2 Using the `$(LoadAddOn` Command

The *tecplot.add* file is a special macro file that is executed at startup time and contains one or more `$(LoadAddOn` commands to load add-ons into Tecplot 360. `$(LoadAddOn` is, in fact, the only macro command allowed in a *tecplot.add* file. The syntax for the `$(LoadAddOn` command is:

```
$(LoadAddOn "libname"
```

where *libname* is the name of the shared object library file or DLL and must be in quotes.

Special rules govern how *libname* name is specified. In all cases the filename extension is omitted.

If you assign *libname* to the basename of the shared object library then Tecplot 360 will do the following:

- **UNIX/Linux/Macintosh** - The shared library to load will come from the file specified by:

```
$TEC_360_2013R1/Lib/Lib+basename+platform-specific-extension
```

Where *platform-specific-extension* is `.sl` for HP platforms and `.so` for all others.

- **Windows** - The add-on *basename.dll* will be searched for in the following order:
 - The directory where the Tecplot 360 executable resides.
 - The Windows system directories.
 - The directories in your PATH environment variable.

If an absolute pathname is used in *libname*, then in Windows, `.dll` is appended, and in Linux `.so` or `.sl` is appended.

4 - 3 Specifying Add-ons under Development

Bugs in an add-on can cause Tecplot 360, and all other add-ons loaded into it, to crash. While you are developing an add-on, we recommend that you keep it isolated, so that other users of Tecplot 360 in your network are not effected by it. The way this is accomplished is somewhat different on Linux or Macintosh platforms than on a Windows operating system.

4 - 3.1 Developing Add-ons on Linux or Macintosh platforms

We recommend that each add-on developer set up a separate "Add-On Development Root Directory." Below this directory, create a separate subdirectory for each add-on. Create a file called *tecdev.add* and put it in the Add-On Development Root Directory. Entries in this *tecdev.add* file must look as follows:

```
$(LoadAddOn "$TECADDONDEVDIR/libmyaddon"
```

As in this example, "myaddon" must be the name of the add-on you are developing. To launch Tecplot 360 so that it will load your add-on that is under development, you use:

```
tec360 -develop
```

This launches Tecplot 360 in a manner such that the *tecdev.add* file in your Add-On Development Root Directory is processed and also sets up the environment variable `TECADDONDEVDIR` so the specific add-on for your platform can be found. If you don't want to load the standard add-ons listed in the main *tecplot.add* file (located in the Tecplot 360 home directory), then include the `-nostdaddons` flag on the command line.

See [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) for more details about developing add-ons.

4 - 3.2 Developing Add-ons on a Windows Operating System

If you use Visual Studio to develop your add-ons in C or C++, we recommend the following setup¹:

1. In Visual Studio, select **Properties** from the **Project** menu.
2. In the navigation pane, select “Debugging”, underneath the “Configuration Properties”.
 - a. Set the “Command” to `$TEC_360_2013R1\bin\tec360.exe`.
 - b. Set the “Command Arguments” to “-loadaddon [path to *project_name.dll*]”.
3. In the navigation pane, go to the Custom Build Step page.
 - a. Set the “Command Line” to “xcopy /d /q \$(TargetPath) \$TEC_360_2013R1\bin”.
 - b. Set “Outputs” to “\$TEC_360_2013R1\\$(TargetPath)”.

This will copy your DLL from `c:\dev\MyAddon\Debug` (or `\Release`) to the `$TEC_360_2013R1\bin` directory after each build, so that when you select **Build** or **Go**, Tecplot 360 will load and run your add-on. See [Chapter 3: “Creating Add-ons on Windows Platforms”](#) for more details about developing add-ons.

1. The setup instruction assume that the add-on **MyAddon** is being developed in `c:\dev\MyAddon`, and that the `$TEC_360_2013R1` environment variable has been set.

Part 2 General Add- on Components

Add-on Initialization and Cleanup

5 - 1 Add-on Initialization

When Tecplot 360 loads an add-on, it makes a call to initialize the add-on via the function `InitTecAddOn`. The only requirement for `InitTecAddOn` is that it call the function `TecUtilAddOnRegister`. The following example shows `TecUtilAddOnRegister` being called from the initialization function.

```
AddOn_pa COMMANDPROCESSORID;
void InitTecAddOn(void)
{
    TecUtilLockOn();
    COMMANDPROCESSORID = TecUtilAddOnRegister(110,
        "My Add-On",
        "V1.0-02/10/06",
        "Acme, Inc.");
    TecUtilLockOff();
}
```

Note this is the only instance where you must use the `TecUtilLockOn` and `TecUtilLockOff` functions. `TecUtilAddOnRegister` registers information which is then accessible to the user via **Help>About Add-Ons**. It also informs Tecplot 360 of the base version of Tecplot (110) on which your add-on was built.

In addition to `TecUtilAddOnRegister`, one or more of the following function calls are typically included in the add-on initialization function:

```
TecUtilImportAddConverter
TecUtilImportAddLoader
TecUtilMenuAddOption
TecUtilCurveRegisterExtCrvFit
```

Each of the preceding function calls provides a different method for end-users to access the add-on via the Tecplot 360 interface.

- `TecUtilMenuAddOption` will add a menu option.
- `TecUtilImportAddConverter` and `TecUtilImportAddLoader` register the add-on as a special type of add-on that is used to load non-Tecplot format data into Tecplot 360. You can access these options from a scrolled list of loaders and converters that is launched when the **File>Data File(s)** dialog is selected. Refer to [Chapter 29: "Creating a Data Converter"](#) or [Chapter 27: "Creating a Data Loader"](#) for additional information.

- `TecUtilCurveRegisterExtCrvFit` registers the add-on as a special type that is used to extend Tecplot 360's XY-plot curve fit capability. You can access these curve fits from the Curve Type option on the Curve page of the **Mapping Style** dialog. Refer to [Chapter 28: "Building Extended Curve Fit Add-ons"](#) for additional information.

At initialization time, an add-on may also elect to launch one or more dialogs immediately in addition to, or in place of, adding menu options to the Tecplot 360 interface.

5 - 2 Add-on Cleanup

When you quit out of Tecplot 360, the following happens:

1. Tecplot 360 queries all add-ons (which have previously made a call to `TecUtilQuitAddQueryCallback`) to determine if their status permits terminating the Tecplot 360 session. A Tecplot 360 shutdown occurs only if all add-ons are ready to be terminated. If an add-on does not want to terminate, then it brings up an error-message dialog informing the user of the problem (and returns FALSE to the call).
2. If all of the add-ons are ready to be terminated, Tecplot 360 will call all of the registered state change functions in add-ons with `StateChange_QuitTecplot`. An add-on can register a state change function by calling `TecUtilStateChangeAddCallback`. When an add-on receives a state change callback with `StateChange_QuitTecplot`, it must assume that the add-on has already given permission to terminate and should only do things such as free previously allocated memory, close open files, etc.



It's best to complete all of your clean up when you get the message `StateChange_QuitTecplot`, rather than waiting for object destructors to get called. Tecplot 360 could be too far along in the shutdown process to use `TecUtil` functions by the time the destructor is called. This is particularly true for global and static objects.

3. Tecplot 360 frees any allocated memory and shuts down.

State Changes From an Add-on

State changes are the method for propagating information when an event occurs. A state change can be triggered by many events. Examples include: loading a data file, changing the color of a mesh plot, creating a new zone, or changing the plot type.

In general, your application should listen for state changes if it needs to take an action based on the state of the Tecplot Engine. For example, if your application displays a dialog which deals with 3D plots, you may want to drop the dialog if the plot type is changed to XY. Alternatively, you may need to update the dialog if a new data set is loaded. Most of the `TecUtil` functions will transmit the necessary state changes automatically.

There are only certain circumstances under which your application will need to send state changes. Refer to [Section 6 - 3 “Sending State Changes”](#) for a list of these situations.



Refer to the sample for examples of working with state changes.

6 - 1 State Change Values

[Table 6 - 1](#) shows the available state change values. Column 1 shows the state change value constants that appear in `GLOBAL.h`.

State Change Value	Explanation	UI Example	API Example
<code>StateChange_VarsAdded</code>	One or more variables were added.	Adding a new variable in the Data> Alter>Specify Equations dialog.	<code>TecUtilDataSetAddVar</code>
<code>StateChange_ZonesAdded</code>	One or more zones were created.	Data>Create Zone>Circular dialog.	<code>TecUtilCreateRectangularZone</code>
<code>StateChange_ZonesDeleted</code>	One or more zones were deleted.	Data>Delete Zones dialog.	<code>TecUtilZoneDelete</code>

Table 6 - 1: State Change Values

State Change Value	Explanation	UI Example	API Example
StateChange_VarsAltered	Values of one or more variables were altered.	Alter variable values in the Data>Alter>Specify Equations dialog.	TecUtilDataAlter
StateChange_NodeMapsAltered	The node map for one or more zones was altered.	Cannot do this from the interface.	TecUtilDataNodeSetByZone
StateChange_DataSetReset	A new data set has been loaded.	File>Load Data File dialog.	TecUtilReadDataSet
StateChange_DataSetFileName	The current data set has been saved to a file.	File>Write Data File dialog.	TecUtilWriteDataSet
StateChange_DataSetTitle	The current data set title has been changed.	Change data set title in Data>Data Set Info dialog.	TecUtilDataSetSetTitle
StateChange_NewLayout	The current layout has been cleared and reset.	File>New Layout menu.	TecUtilNewLayout
StateChange_NewTopFrame	A new frame has become the current frame.	Frame>Order Frames dialog.	TecUtilFrameCreateNew
StateChange_FrameDeleted	A frame was deleted.	Frame>Delete Current Frame menu.	TecUtilFrameDeleteTop
StateChange_PageDeleted	A page was deleted.	N/A	
StateChange_NewTopPage	The current page has changed.	N/A	
StateChange_Style	The style of the plot has been altered.	Zone Style>Mapping Style dialogs.	tecplot::toolbox::StyleValue
StateChange_Text	One or more text elements have changed.	Adding, removing, or modifying text.	Explicitly by calling TecUtilStateChanged with StateChange_Text.
StateChange_Geom	One or more geometry elements have changed.	Adding, removing, or modifying geometries.	Explicitly by calling TecUtilStateChanged with StateChanged_Geom.
StateChange_LineMapAssignment	An X-Y mapping definition has been altered (includes zone and axis information).	Definitions page of the Mapping Style dialog.	TecUtilLineMapSetAssignment
StateChange_ContourLevels	The contour levels have been altered.	Contour Levels.	TecUtilContourLevelAdd
StateChange_ZoneName	The name of a zone has been altered.	Rename a zone in the Data>Data Set Info dialog.	TecUtilZoneRename
StateChange_VarName	The name of a variable has been altered.	Rename a variable in the Data>Data Set Info dialog.	TecUtilVarRename
StateChange_LineMapName	The name of a line mapping has been altered.	Rename a line map in an XY Line plot.	TecUtilLineMapSetName

Table 6 - 1: State Change Values

State Change Value	Explanation	UI Example	API Example
StateChange_LineMapAddDeleteOrReorder	The set of existing line mappings has been altered.	Mapping Style dialog, [Create Map] button.	TecUtilLineMapDelete
StateChange_View	The view of the plot has been altered (usually a translate, scale, or fit action).	View>Translate-Magnify dialog.	TecUtilViewTranslate
StateChange_ColorMap	The color mapping has been altered.	Options>Color Map dialog.	TecUtilColorMapResetToFactory
StateChange_ContourVar	The contour variable has been reassigned.	Plot>Contour Details dialog.	TecUtilContourSetVariable
StateChange_Streamtrace	The set of streamtraces, a termination line, or the streamtrace delta time has been altered.	Plot>Streamtrace Details dialog.	TecUtilStreamtraceAdd
StateChange_NewAxisVariables	The axis variables have been reassigned.	Plot>Assign XYZ dialog.	tecplot::toolbox::StyleValue
StateChange_MouseModeUpdate	A new mouse mode (tool) has been selected.	Select a new mouse mode (tool) in the sidebar.	TecUtilPickSetMouseMode
StateChange_PickListCleared	All picked objects are unpicked. ^a	Click on the paper in the workspace.	TecUtilPickDeselectAll
StateChange_PickListGroupSelect	A group of objects has been added to the pick list.	Draw a box around a group of objects with the selector or adjustor tool.	TecUtilPickAddAll
StateChange_PickListSingleSelect	A single object has been added to or removed from the pick list. ^a	Select an object with the selector or adjustor tool.	TecUtilPickAtPosition
StateChange_PickListStyle	An action has been performed on all of the objects in the pick list. ^a	Quick Edit dialog.	TecUtilPickEdit
StateChange_ModalDialogLaunch	A modal dialog has been launched.	Options>Ruler/Grid dialog.	TecUtilDialogMessageBox
StateChange_ModalDialogDismiss	A modal dialog has been dismissed.	Options>Ruler/Grid dialog.	TecUtilDialogMessageBox
StateChange_CompleteReset	Anything could have happened.	File>Open Layout dialog.	TecUtilOpenLayout
StateChange_UnsuspendInterface	Graphics have been turned back on.	Execute the macro command \$!DRAWGRAPHICS ON.	TecUtilDrawGraphics(TRUE);
StateChange_SuspendInterface	Graphics have been turned back off.	Execute the macro command \$!DRAWGRAPHICS OFF.	TecUtilDrawGraphics(FALSE);

Table 6 - 1: State Change Values

State Change Value	Explanation	UI Example	API Example
StateChange_DataSetLockOn	The dataset attached to the active frame has been locked.	Cannot be done via user interface.	TecUtilDataSetLockOn
StateChange_DataSetLockOff	The dataset attached to the active frame has been unlocked.	Cannot be done via user interface.	TecUtilDataSetLockOff
StateChange_DrawingInterrupted	The user has interrupted the drawing.	User clicks with the mouse in the work area.	TecUtilInterrupt
StateChange_QuitTecplot	The Tecplot Engine is about to exit.	File/Exit menu.	
StateChange_AuxDataAdded	Auxiliary data was added.	Cannot do this via the user interface.	tecplot::toolbox::AuxData
StateChange_AuxDataDeleted	Auxiliary data was deleted.	Cannot do this via the user interface.	tecplot::toolbox::AuxData
StateChange_AuxDataAltered	Auxiliary data was altered.	Cannot do this via the user interface.	tecplot::toolbox::AuxData
StateChange_PrintPreviewLaunch	When entering "Print Preview" mode.	File>Print Preview...	N/A
StateChange_PrintPreviewDismiss	When exiting "Print Preview" mode.	Pressing [Close] button in "Print Preview" mode.	N/A
StateChange_VarsDeleted	One or more variables were deleted.	Data>Delete>Variable...	TecUtilDataSetDeleteVar
StateChange_TecplotIsInitialized	When running in interactive mode this state change is broadcast when initialized. In batch mode this message is not broadcast.	N/A	N/A
StateChange_ImageExported	Broadcast after each frame of an image is exported.	Exporting single images and movies.	TecUtilExportNextFrame
StateChange_VariableLockOn	Indicates that a variable has been locked for either delete or value modification.	N/A	TecUtilVariableLockOn
StateChange_VariableLockOff	Indicates that a variable has been locked from either delete or value modification.	N/A	TecUtilVariableLockOff

Table 6 - 1: State Change Values

- a. See [Chapter 22: "Working with Picked Objects"](#) for more information on picked objects and the pick list.

6 - 2 Listening for State Changes

In order to listen for state changes, an object must register itself as a state change listener. This may be done via



Example 6: Adding a Dialog that is Updated When Contour Items Change

```
#include "StateChangeListenerInterface.h"
class ContourDialogSimple :
    public tecplot::toolbox::StateChangeListenerInterface
{
public:
    void launchDialog();
    void dropDialog();
private:
    // Update the controls on the dialog
    void updateDialog();

    // Implementation of StateChangeListenerInterface
    // NOTE: tbx is an alias for tecplot::toolbox.
    void stateChanged(const tbx::StateChangeEventInterface&
stateChangeEvent,
                    tbx::StateChangeNotifierInterface&    source);
};

void ContourDialogSimple::launchDialog()
{
    // Begin listening for state changes when the dialog is launched.
    StateChangeEventObservable::getInstance().addListener(this);

    // TODO: add logic to launch the dialog
    updateDialog();
}

void ContourDialogSimple::dropDialog()
{
    // Remove the state change listener when the dialog drops
    // since we don't need to listen to state changes anymore
    StateChangeEventObservable::getInstance().removeListener(this);
    // TODO: add logic to drop the dialog
}

void ContourDialogSimple::updateDialog()
{
    // TODO: update controls on the dialog
}

// stateChanged is called whenever a state change event occurs.
// This listener inspects the stateChangeEvent and only performs
// and action in response to certain state change events.
void ContourDialogSimple::stateChanged(const
tbx::StateChangeEventInterface& stateChangeEvent,

tbx::StateChangeNotifierInterface&    source)
{
    VERIFY(&source == &StateChangeEventObservable::getInstance());
    switch ( stateChangeEvent.getStateChange() )
    {
        {
            case StateChange_ContourLevels :
```

```

        case StateChange_ContourVar      :
            updateDialog();
            break;
        case StateChange_QuitTecplot :
            dropDialog();
            break;
        default:
            break;
    }
}

```

6 - 2.1 Supplemental Information

Many state change events include supplemental information which gives more specific information about the particular state change event. When your listener's `stateChanged` method is called, a `StateChangeEventInterface` is included. This object contains all available supplemental information.

The state changes that have supplemental information that can be queried are:

State Change	Supplemental Information	StateChangeEventInterface Method
StateChange_VarsAltered	VarSet, ZoneSet (optional), Index (optional), Frame unique ID	getVarSet(), getZoneSet(), getIndex(), getFrameUniqueID()
StateChange_VarsAdded	VarSet, Frame unique ID	getVarSet(), getFrameUniqueID()
StateChange_ZoneDeleted	ZoneSet, Dataset unique ID	getZoneSet(), getDataSetUniqueID()
StateChange_NodeMapAltered	ZoneSet, Dataset unique ID	getZoneSet(), getDataSetUniqueID()
StateChange_MouseModeUpdate	Enum for the new mouse mode (MouseButtonMode_e)	getArbEnum()
StateChange_Style	Style parameters P1, P2, P3, P4, P5, P6 (P2-P6 are optional), Page unique ID for page commands, Frame unique ID for frame commands	getStyleParam(), getStyleParams(), getOffset1, getOffset2(), getFieldmapSet(), getLinemapSet(), getPageUniqueID(), getFrameUniqueID()
StateChange_View	Enum for view action (View_e), Frame unique ID	getArbEnum(), getFrameUniqueID()
StateChange_Streamtrace	Enum for Streamtrace action (Streamtrace_e), Frame unique ID	getArbEnum(), getFrameUniqueID()
StateChange_AuxDataAltered	Enum for Aux. Location (AuxDataLocation_e), Zone if location is AuxDataLocation_Zone, Page unique ID for page aux. data, Frame unique ID for frame aux data, Dataset unique ID for dataset aux data	getArbEnum(), getPageUniqueID(), getFrameUniqueID(), getDataSetUniqueID()

Table 6 - 2: State changes and supplemental information.

State Change	Supplemental Information	StateChangeEventInterface Method
StateChange_AuxDataAdded	Enum for Auxiliary Location (AuxDataLocation_e), Zone if location is AuxDataLocation_Zone, Page unique ID for page aux. data, Frame unique ID for frame aux data, Dataset unique ID for dataset aux data	getArbEnum(), getPageUniqueID(), getFrameUniqueID(), getDataSetUniqueID()
StateChange_AuxDataDeleted	Enum for Auxiliary Location (AuxDataLocation_e), Zone if location is AuxDataLocation_Zone Page unique ID for page aux. data, Frame unique ID for frame aux data, Dataset unique ID for dataset aux data	getArbEnum(), getPageUniqueID(), getFrameUniqueID(), getDataSetUniqueID()
StateChange_VariableLockOn	Var, VarLockMode:ArbParam_t, LockOwner:Name Dataset unique ID	getName(), getDataSetUniqueID()
StateChange_VariableLockOff	Var, LockOwner:Name Dataset unique ID	getName(), getDataSetUniqueID()
StateChange_DataSetLockOn	LockOwner:Name Dataset unique ID	getName(), getDataSetUniqueID()
StateChange_DataSetLockOff	LockOwner:Name Dataset unique ID	getName(), getDataSetUniqueID()
StateChange_VarsDeleted	VarSet	getVarSet()
StateChange_QuitTecplot	The Tecplot Engine exit code.	getIndex()

Table 6 - 2: State changes and supplemental information.

Note that not all supplemental information may be available all the time. In some cases this information may not be supplied at all. If information is not supplied, then you must assume the worst case. For example, on a StateChange_VarsAltered, if the ZoneSet is not supplied you must assume that the variables were altered in all zones. For the VarsAltered state change, you can assume that the VarSet is supplied.

Only certain state changes will probably be of interest to your application. In the example above, only state changes which involve the contour variable or the contour levels are monitored.

For an example where supplemental information is used, consider the following case where the contour dialog instead wants to update only when the contour variable is altered:

```
void ContourDialogSimple::stateChanged(const
tbx::StateChangeEventInterface& stateChangeEvent,
                                     tbx::StateChangeNotifierInterface&
source)
{
    StateChange_e stateChange = stateChangeEvent.getStateChange();
    if ( stateChange == StateChange_VarsAltered )
    {
        // If the contour variable was altered, update the dialog
        EntIndex_t contourVarNum = TecUtilVarGetNumByAssignment('C');
    }
}
```

```

        if ( stateChangeEvent.hasVarSet() &&
            stateChangeEvent.getVarSet().isMember(contourVarNum) )
        {
            updateDialog();
        }
    }
    else if ( StateChange_QuitTecplot )
        dropDialog();
}

```

6 - 2.2 Coding Rules for State Change Listeners

When your state change listener is notified of a state change, it is almost always the case that either the Tecplot Engine or your application is in the middle of a sequence of tasks. Thus, we strongly recommend that you do not include code in your listener that generates state changes. All TecUtil functions and class methods that query state information are acceptable, as those requests do not cause the state to change. Executing functions such as TecUtilCreateRectangularZone is not recommended since they modify the state.

If you must react to a state change event by making a TecUtil function call or class method call that changes the state in the Tecplot Engine, it is recommended that you do this on idle. Your application may use TecUtilOnIdleQueueAddCallback

6 - 3 Sending State Changes

There are only certain circumstances under which your application will need to send state changes to the Tecplot Engine. Most of the TecUtil functions will transmit the necessary state changes automatically.

For example, when an application calls TecUtilZoneDelete, the StateChange_ZonesDeleted state change is automatically transmitted. Currently, your application must send state changes under the following circumstances:

Circumstance	Relevant State Change Value	Supplemental Information Supplied to the Tecplot Engine
After a variable has been added and subsequently modified.	StateChange_VarsAdded	None
After a variable has been modified.	StateChange_VarsAltered	Set of affected variables. Index of value changed. Set of affected zones.
After the node map has been modified.	StateChange_NodeMapsAltered	Set of affected zones
After TecUtilDataSetAddZone has been called and the field data set has been subsequently modified.	StateChange_ZonesAdded	Set of affected zones
After adding, removing, or modifying one or more text elements.	StateChange_Text	None
After adding, removing or modifying one or more geometry elements.	StateChange_Geom	None

Table 6 - 3: State changes that applications are allowed to send.

When an application submits a state change with `StateChange_VarsAltered`, you may supply the set of variables altered (required), the set of zones in which those variables were altered, and the index of the point that was altered (if only one value was altered).



Example 7: Broadcasting a State Change

The following example illustrates how to broadcast a state change. The example has altered the 3rd variable in zones 5 and 6 at offset 10 and needs to broadcast the state change:

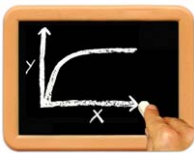
NOTE: Error checking has been omitted for clarity.

```
Set zoneSet;
Set varSet(3);
zoneSet.add(5);
zoneSet.add(6);

StateChangeEventObservable::sendStateChange(StateChange_VarsAltered,
                                             zoneSet,
                                             varSet,
                                             10);
```

As discussed in the previous sections, the state change listeners can choose to use any or all of the supplemental information. Listeners must also know how to handle situations where the supplemental information is not supplied (assume worst case).

NOTE: Only the state change notifications listed in [Table 6 - 3](#) may explicitly originate from your application.



Example 8: Broadcasting a State Change

The following example illustrates how to broadcast a state change. The example modifies an add-on that alters the 3rd variable in zones 5 and 6 at offset 10. Because the add-on alters data, a state change must be broadcast:

```
{
    ArgList_pa          ArgList;
    Set_pa              ZoneList;
    Set_pa              VarList;
    ArgList = TecUtilArgListAlloc;
    ZoneList = TecUtilSetAlloc(FALSE);
    VarList = TecUtilSetAlloc(FALSE);
    TecUtilSetAddMember(ZoneList,5,FALSE);
    TecUtilSetAddMember(ZoneList,6,FALSE);
```

```

    TecUtilSetAddMember(VarList,3,FALSE);
    TecUtilArgListAppendInt(ArgList,SV_STATECHANGE,
        (LgIndex_t)StateChange_VarsAltered);
    TecUtilArgListAppendSet(ArgList,SV_ZONELIST,ZoneList);
    TecUtilArgListAppendSet(ArgList,SV_VARLIST,VarList);
    TecUtilArgListAppendInt(ArgList,SV_INDEX,10);
    TecUtilStateChangedX(ArgList);
    TecUtilSetDealloc(&ZoneList);
    TecUtilSetDealloc(&VarList);
    TecUtilArgListDealloc(&ArgList);
}

```

Note: Error checking has been omitted for clarity.

As discussed in the previous sections, the state change listeners can choose to use any/all of the supplemental information they desire. Listeners must also know how to handle situations where the supplemental information is not supplied (assume worst case).

To see examples of the use of `TecUtilStateChanged` or `TecUtilStateChangedX`, please refer to the [ADK Reference Manual](#). However, only the state change notifications listed in [Table 6 - 3](#) may explicitly originate from your add-on.

Locking and Unlocking Tecplot 360

Most classic function calls (i.e. TecUtilXXX functions) require that the Tecplot Engine be locked. The only exceptions to this are for the lock functions themselves and for most query type functions, such as TecUtilFrameGetPlotType.

You should call TecUtilLockStart at the beginning of any function that calls TecUtil functions and call TecUtilLockFinish at the end. Note that you must pass your add-on ID to these functions.



Example 9: Switching the Frame Plot Type to 3D Cartesian

The following sample code illustrates switching the plot type to 3D Cartesian. Because this involves a call to a classic TecUtil function, you must also use the lock functions.

```
void SwitchFrameTo3D(void)
{
    TecUtilLockStart();
    TecUtilFrameSetPlotType(PlotType_Cartesian3D);
    TecUtilLockFinish();
}
```

The following table lists the functions that control or monitor locking and unlocking in the Tecplot Engine:

Function Prototype	Notes
<code>void TecUtilLockStart()</code>	Locks the Tecplot Engine. You may call <code>TecUtilLockStart</code> any number of times, as long as each call is matched with a call to <code>TecUtilLockFinish</code> .
<code>void TecUtilLockFinish()</code>	Unlocks the Tecplot Engine. You must have exactly one call to <code>TecUtilLockFinish</code> for each call to <code>TecUtilLockStart</code> .
<code>Boolean_t TecUtilLockIsOn(void)</code>	Returns TRUE if the Tecplot Engine is currently locked.
<code>int TecUtilLockGetCount(void)</code>	Returns the number of levels of locking that are currently active in the Tecplot Engine.

Table 7 - 1: Tecplot Engine lock functions

Working with Multi-threading

Incorporating multi-threading into your add-on or application allows you to take advantage of multiple processors (when they are present) thereby increasing the efficiency of your application. Threads are present under any of the following scenarios:

- **Receiving a load-on-demand callback.** Refer to [Section 23 - 5 “Load-on-demand”](#) for information on working with load-on-demand callbacks.
- **Using threading associated with your own add-on or application.** Note: You are not required to use the threading functions provided by the Tecplot Engine; they are provided for your convenience. You can use your own threading code. However, if you are creating an application from scratch or a data loader for the Tecplot Engine, you may prefer to use these functions.
- **Adding threading via the Tecplot Engine API.** Threading can be incorporated using TecUtil functions with of the following methods: the thread pool, a mutex, or a condition variable. Refer to the remainder of this chapter for information regarding the remaining scenarios.

While you are in a thread, you can only call functions that are thread safe. Functions that are thread safe are labeled as such in the .

8 - 1 Thread pool

If your add-on or application performs complex calculations, we recommend using the thread pool in order to decrease the calculation processing time. To incorporate threading into your add-on or application, via the Tecplot Engine thread pool use the following basic steps:

1. Call `TecUtilThreadPoolJobAlloc` to obtain a job control variable. The job control variable is used to associate jobs submitted to the thread pool with one another.
2. Call `TecUtilThreadPoolAddJob` to create a job associated with the job control variable. `TecUtilThreadPoolAddJob` takes a callback for the function to be executed by the thread pool.
3. Call `TecUtilThreadPoolWait` to block the current thread until all jobs have finished.
4. Call `TecUtilThreadPoolJobDealloc` to deallocate the job control once it is no longer needed.

Use `TecUtilThreadPoolGetNumConcurrentJobs` to determine the number of jobs that can be run concurrently within the thread pool. This number is limited first by the number specified by entering `$!LIMITS`

MAXAVAILABLEPROCESSORS = xxx your *tecplot.cfg* file. If MAXAVAILABLEPROCESSORS is not set or is set to "0", the number of available threads is determined by your system. However, the number cannot be greater than 8 for Tecplot 360 or 1 for Tecplot Focus.

8 - 2 Thread Mutex

Use a mutex to prevent multiple threads from changing the same variable or function simultaneously. The mutex should be allocated in a thread-safe portion of the code. Perform the following basic steps to use a thread mutex:

1. Call `TecUtilThreadMutexAlloc` to obtain a mutex variable.
2. Call `TecUtilThreadMutexLock` to lock the mutex allocated above.
3. Perform a function involving global data.
4. Call `TecUtilThreadMutexUnlock` to unlock the mutex.
5. Call `TecUtilThreadMutexDealloc` to deallocate the mutex once it is no longer needed.

8 - 3 Condition Variables

Use a condition variable to activate a thread when a user-specified condition is met. The following functions use condition variables and threads:

- `TecUtilThreadBroadcastCondition`
- `TecUtilThreadConditionAlloc`
- `TecUtilThreadConditionDealloc`
- `TecUtilThreadSignalCondition`
- `TecUtilThreadTimedWaitForCondition`
- `TecUtilThreadWaitForCondition`

Part 3 Data

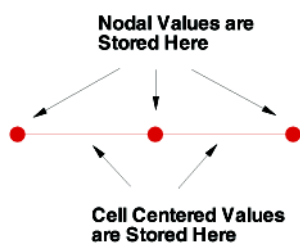
Data Structure

There are three primary classes of data structure: [Ordered Data](#), [Finite Element Data](#) and [Face Neighbors](#).

9 - 1 Ordered Data

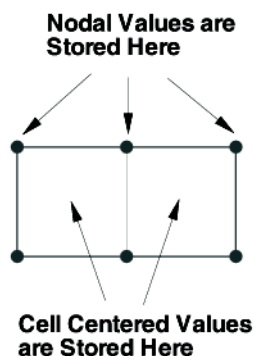
Ordered data is defined by one, two, or three-dimensional logical arrays, dimensioned by IMAX, JMAX, and KMAX. These arrays define the interconnections between nodes and cells. The variables can be either nodal or cell-centered. Nodal variables are stored at the nodes; cell-centered values are stored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

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



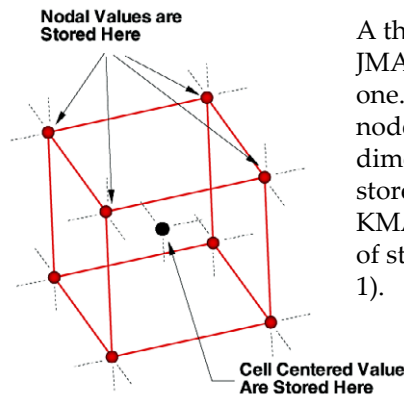
A single dimensional array where either IMAX, JMAX or KMAX is greater than or equal to one, and the others are equal to one. For nodal data, the number of stored values is equal to $IMAX * JMAX * KMAX$. For cell-centered I-ordered data (where IMAX is greater than one, and JMAX and KMAX are equal to one),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

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



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

- **Three-dimensional Ordered Data (IJK-ordered)**



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

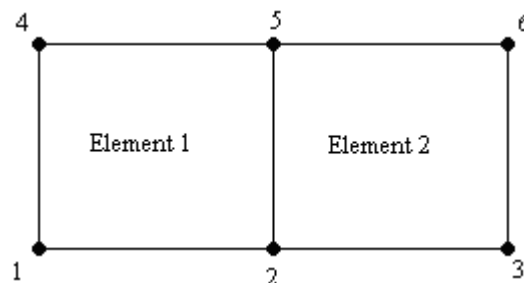
9 - 1.1 Face Neighbors for Ordered Data

You may also need to specify how elements are laid out with respect to one another. If each face in each element has only one neighbor and you do not need to specify how elements relate across zones, you do not need to provide face neighbor information. Otherwise, refer to [Section 9 - 3 "Face Neighbors"](#) for detailed information on working with face neighbors.

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

The number of nodes required for each element is implied by your zone type. For example, if you have a finite element quadrilateral zone, you will have four nodes defined for each element. Likewise, you must provide eight numbers for each cell in a BRICK zone, and three numbers for each element in a TRIANGLE zone. If you have a cell that has a smaller number of nodes than that required by your zone type, simply repeat a node number. For example, if you are working with a finite element quadrilateral zone and you would like to create a triangular element, simply repeat a node in the list (e.g., 1,4,5,5).

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



The connectivity list for this example would appear as follows:

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



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

9 - 2 Finite Element Data

There are two classes of finite element data in the Tecplot 360: classic elements and polyhedral elements. Classic elements are cell-based finite elements, where each element in a zone has the same number of nodes. The following cell-based element types are available: triangular, quadrilateral, tetrahedron and brick. Polyhedral elements are face-based elements, where each element in a zone can have a variable number of nodes and faces. Refer to [“Face Neighbors for Classic Finite Element Data”](#) on page 67 and [“FaceNodeCounts and FaceNodes”](#) on page 68 for details.

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

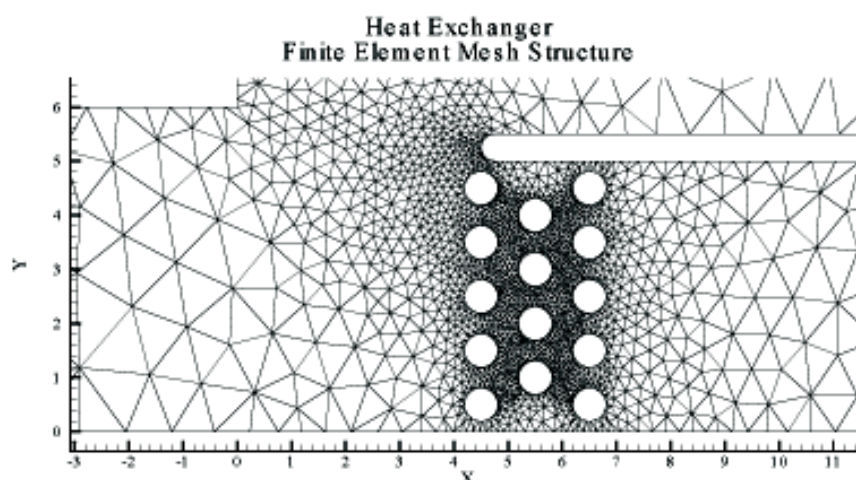


Figure 9-1. This figure shows finite element data used to model a complex boundary. This plot file, *feexchng.plt*, is located in your Tecplot 360 distribution under the *examples/2D* subdirectory.

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

- **Line data** is a set of line segments defining a 2D or 3D line. Unlike I-ordered data, a single finite element line zone may consist of multiple disconnected sections. The values of the variables at each data point (node) are entered in the data file similarly to I-ordered data, where the nodes are numbered with the I-index. This data is followed by another set of data defining connections between nodes. This second section is often referred to as the connectivity list. All elements are lines consisting of two nodes, specified in the connectivity list.
- **Surface data** is a set of triangular, quadrilateral, or polygonal elements defining a 2D field or a 3D surface. When using polygonal elements, the number of sides may vary from element to element. In finite element surface data, you can choose (by zone) to arrange your data in three point (triangle), four point (quadrilateral), or variable-point (polygonal) elements. The number of points per node and their arrangement are determined by the element type of the zone. If a mixture of quadrilaterals and triangles is necessary, you may repeat a node in the quadrilateral element type to create a triangle, or you may use polygonal elements.

- **Volume data** is a set of tetrahedral, brick or polyhedral elements defining a 3D volume field. When using polyhedral elements, the number of sides may vary from element to element. Finite element volume cells may contain four points (tetrahedron), eight points (brick), or variable points (polyhedral). The figure below shows the arrangement of the nodes for tetrahedral and brick elements. The connectivity arrangement for polyhedral data is governed by the method in which the polyhedral facemap data is supplied.

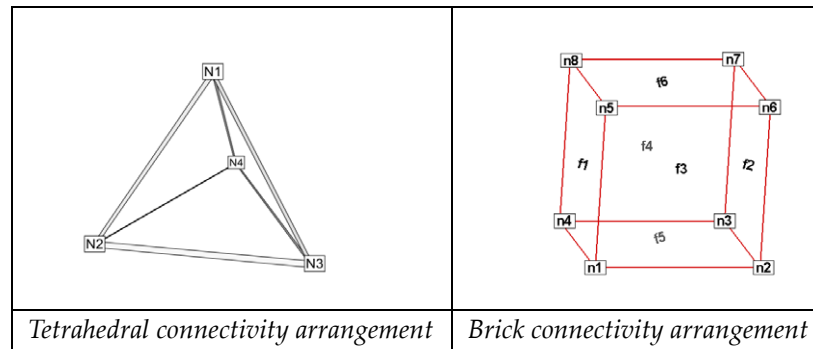


Figure 9-2. Connectivity arrangements for FE-volume datasets

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

[Section 4 - 5 “Finite Element Data”](#) in the [Data Format Guide](#) provides detailed information about how to format your FE data in Tecplot’s data file format.

9 - 2.1 Indexing Nodal Finite Element Data

For nodal finite element data, there is a one-to-one correspondence between the nodal values supplied in the data file and the index you use to access these values. Thus to access the 5th nodal value for the 5th data point, use an index of 5.

9 - 2.2 Indexing Cell-centered Finite Element Data

For cell-centered finite element data there is a one-to-one correspondence between the cell centered values supplied in the data file and index you use to access these values. For example, to access the cell centered value for the 5th cell in the connectivity list, use an index of 5.

Classic finite elements are cell-based elements, where each element in a zone has the same number of nodes.

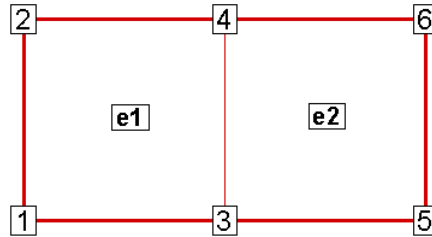
Connectivity List for Classic Finite Elements

The connectivity list is used to define the node numbers contained in each element of a classic FE zone. A collection of nodes is used to create an element or cell. The number of nodes in each element is defined by the zone type, e.g., quadrilateral zones have four nodes in each element, brick zones have eight nodes in each element, etc.

The number of values in the connectivity list is equal to the number of elements in the zone multiplied by the number of nodes in the element type of the zone. For example, a triangular zone with five elements will have 15 values in its connectivity list. The first three values define the nodes in element 1, the second three values define the nodes in element 2 and so on.

When providing the connectivity list, please keep in mind the following guidelines:

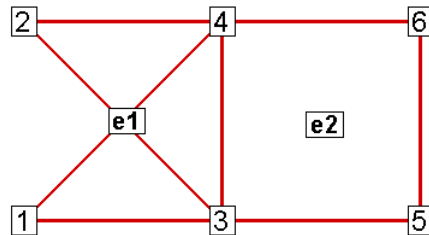
- For nodal data, the node numbering is implicitly defined by the order in which the data values are supplied (i.e. the first value supplied is for node 1, followed by the value for node 2, and so on). The node numbers must be provided in order, either clockwise or counter-clockwise. Otherwise, the elements may appear misshapen. For example, given the node numbering for two elements as defined in the following picture,



the connectivity list would be:

```
1,3,4,2,
3,5,6,4
```

However, if the first two numbers were switched, the zone would appear as follows:



- 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.

Face Neighbors for Classic Finite Element Data

In addition to specifying the nodes that compose each element, you may also need specify how elements are connected to one another. If each face in each element has only one neighbor and you do not need to specify how elements relate across zones, you do not need to provide face neighbor information. Otherwise, refer to the [Data Format Guide](#) for detailed information on working with face neighbors.

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.

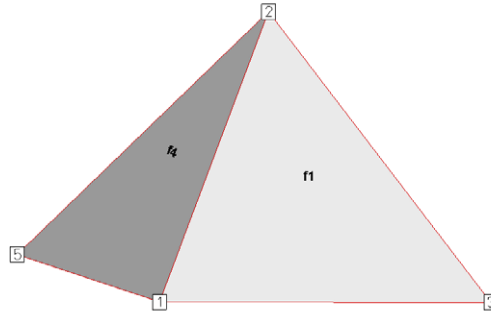


Figure 9-3. A simple pyramid. The remaining triangular faces are Faces 2 and 3. The bottom rectangular face is Face 5. Node 4 is obscured from view.

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

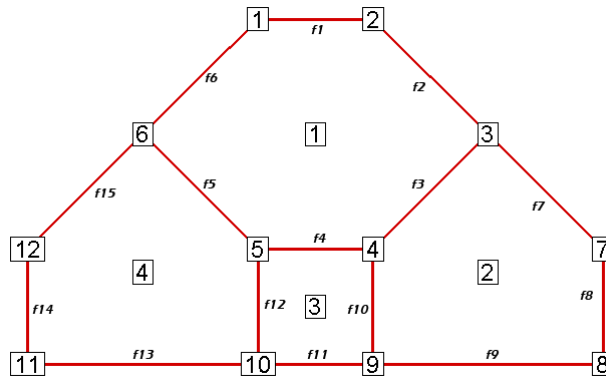
```

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

```

FaceRightElems and FaceLeftElems

After specifying the face map data (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 preceding plot, the face neighbors are as follows:

Face Number	Right Neighboring Element	Left Neighboring Element
Face 1	1	0
Face 2	1	0

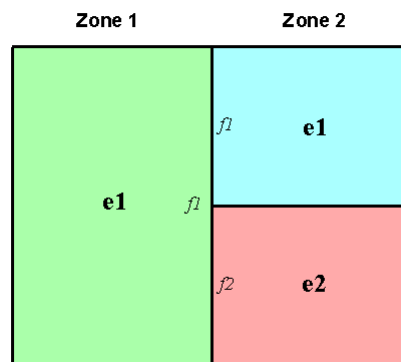
Face Number	Right Neighboring Element	Left Neighboring Element
Face 3	1	2
Face 4	1	3
Face 5	1	4
Face 6	1	0
Face 7	2	0
Face 8	2	0
Face 9	2	0
Face 10	2	3
Face 11	3	0
Face 12	3	4
Face 13	4	0
Face 14	4	0
Face 15	4	0

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

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 this figure, 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

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.

9 - 3 Face Neighbors

Use face neighbors to specify connections between zones (global connections) or connections within zones (local connections) for ordered or classic finite-element data¹. Face neighbor connections are used 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.

The nature of the data arranged in the Face Neighbor Connections list depends upon the Face Neighbor Mode, described in the following table. To connect the cells along one edge to cells on another edge of the same zone, use one of the "local" types. To connect cells of one zone to cells of another zone or zones, use one of the "global" types. If the points of the cells are exactly aligned with the neighboring cell points, use "one-to-one". If even one cell face is neighbor to two or more other cell faces, use one-to-many".

Face Neighbor Mode	Number of Values	Order of Data in the Face Neighbor Connections List
Local One-to-one	3	The cell number in the current zone.
		The number of the cell face in the current zone.
		The cell number of the neighbor cell in the current zone.

Table 9 - 2: Face Neighbor Modes

1. Refer to ["Facemap Data"](#) on page 156 for details on defining connections for finite-element polyhedral data.

Face Neighbor Mode	Number of Values	Order of Data in the Face Neighbor Connections List
Local One-to-many	nz^2+4	The cell number in the current zone.
		The number of the cell face in the current zone.
		Face obscuration flag (zero for face partially obscured, one for face entirely obscured).
		The number of neighboring cells for the “one-to-many” options.
		The cell number of the neighbor cell in the current zone, as follows: cell number 1, cell number 2...cell number n (where n is the number of cell numbers).
Global One-to-one	4	The cell number in the current zone
		The number of the cell faces in the current zone
		The remote zone number
		The cell number of the neighboring cell in the remote zone
Global One-to-many	$2*nz^2+4$	The cell number in the current zone
		The number of the cell faces in the current zone
		Face obscuration flag (zero for face partially obscured, one for face entirely obscured)
		The number of neighboring cells for the “one-to-many” options
		The remote zone number (z)
		The cell number in the remote zone of the n th neighboring cell in the global one-to-many list (c)
		The list of neighboring cells and their zones as follows: neighboring cell 1, remote zone 1...neighboring cell n , remote zone n (where n is the number of neighboring cells).

Table 9 - 2: Face Neighbor Modes

a. nz is equal to the number of neighboring cells for the “one to many” options.

The combination of cell and face numbers in the current zone must be unique; multiple entries are not allowed. The face numbers for cells in the various zone types are defined in [Figure 9-4](#).

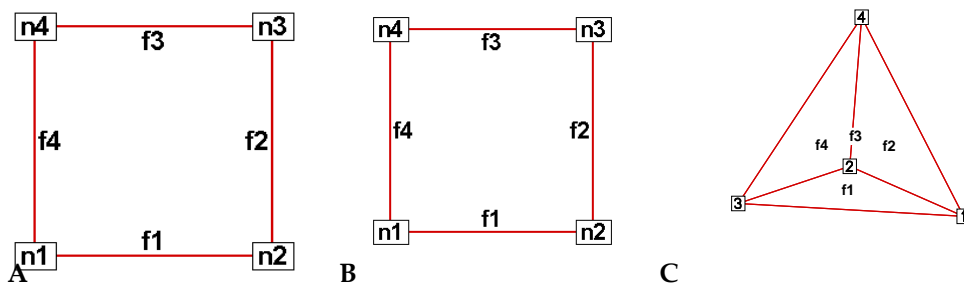


Figure 9-4. **A:** Example of node and face numbering for an fe-brick cell or IJK-ordered cell. **B:** Example of node and face numbering for an IJ-ordered cell. **C:** Example of tetrahedron node and face numbering.

A connection must be specified for two matching cell faces to be effective. For example, for data with a Face Neighbor Mode of global one-to-one, 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 the Tecplot Engine 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 improve the continuity of contours and streamtraces at zone boundaries.

Accessing Field Data

There are several ways to query and set field data. Each method has advantages and disadvantages with respect to speed and ease-of-use.

10 - 1 Data Access Methods

There are several methods to access data loaded into the Tecplot Engine. We recommend using [By Reference \(Array\)](#). However, there are some situations where a different method is preferred.

Preferred	Method	Efficiency	Notes
*	By Reference (Array)	High	Use when the data source is large and/or data can be buffered to send multiple values at a time.
	By Reference (Single Value)	Medium	Use when you are working with the LoadValueOnDemand loader approach (as only one value is set at a time).
	By Specifying Zone-variable	Low	Use only for small data sets.

When accessing data stored in the Tecplot Engine, be aware that all data access via the `TecUtil` layer is “1-based.” For example, the first value in the data set is at an offset of 1. In addition, all data is accessed as a contiguous array of values using a single `LgIndex_t` offset regardless of a zone's dimensions.

10 - 1.1 By Reference (Array)

This method requires that you set up a `FieldData_pa` pointer to reference the data. However, it is the fastest method because it submits many values at once, while offering error checking. Unlike `TecUtilDataValueSetByRef`, it does not offer any parameter conversion.



Example 10: Accessing Data By Reference (Array)

Send data in chunks to the second variable of zone 5:

```
#define CHUNK_SIZE 1024
double Buffer[CHUNK_SIZE]; /* array type MUST match field data type */
FieldData_pa FD = TecUtilDataValueGetWritableRef(5,2);
LgIndex_t NumValuesToSend = TecUtilDataValueGetCountByRef(FD);
LgIndex_t NumIterations = NumValuesToSend / CHUNK_SIZE;
LgIndex_t NumRemainingValues = NumValuesToSend % CHUNK_SIZE;

LgIndex_t Iteration;
for (Iteration = 0; Iteration < NumIterations; Iteration++)
{
    ... fetch up to CHUNK_SIZE data from source

    /* send up to 1024 values of data at a time */
    LgIndex_t DestOffset = Iteration*CHUNK_SIZE + 1;
    TecUtilDataValueArraySetByRef(FD,
                                   DestOffset,
                                   CHUNK_SIZE,
                                   (void *)Buffer);
}

if (NumRemainingValues != 0)
{
    ... fetch the remaining values from source

    /* send the remaining values of data at once */
    LgIndex_t DestOffset = NumIteration*CHUNK_SIZE
    TecUtilDataValueArraySetByRef(FD,
                                   DestOffset,
                                   NumRemainingValues,
                                   (void *)Buffer);
}
```

For brevity, we assumed that the field data was of type double in the previous example. The field data type can be determined by calling:

```
FieldDataType_e fd_type = TecUtilDataValueGetRefType(FD);
```

10 - 1.2 By Reference (Single Value)

You can assign or get the value of a field variable at a specific position using `TecUtilDataValueSetByRef` or `TecUtilDataValueGetByRef`. This method requires that you set up a `FieldData_pa` pointer to reference the data, but it is much faster than [By Specifying Zone-variable](#), while still offering error checking and parameter conversion.

These functions use the double data type, regardless of the field data type of the zone. If the field data type is not double, then these functions will perform an appropriate conversion for you. If needed, you may find out the type of the field data by calling:

```
FieldDataType_e fd_type = TecUtilDataValueGetRefType(FD);
```

If high performance is not essential, we recommend using either this method or [TecUtilDataValueSetByZoneVar & TecUtilDataValueGetByZoneVar](#) to access field data.



Example 11: Accessing Data By Reference (Single Value)

Add 1 to the first data point of the second variable of zone 5.

```
double value;
FieldData_pa FD = TecUtilDataValueGetWritableRef(5,2);
/* Zone 5, Variable 2 */
/* You can now use FD to get/set the data. */
value = TecUtilDataValueGetByRef(FD,1);
value += 1.0;
TecUtilDataValueSetByRef(FD,1,value);
/* No need to free FD. */
```

Note that in order to modify a value in the field data we fetched a writable field data reference. If we were only inspecting values then it would be more efficient to acquire a readable reference via, `TecUtilDataValueGetReadableRef`:



Example 12: Accessing Data By Reference and Changing Values

Set all the interior nodal values for variable V in zone Z to 99.0. Do this only if zone Z is an IJK-ordered zone and variable V is node located.

```
if ((TecUtilZoneGetType(Z) == ZoneType_Ordered) &&
    (TecUtilDataValueGetLocation(Z,V) == ValueLocation_Nodal))
{
    TecUtilZoneGetIJK(Z,
                      &IMax,
                      &JMax,
                      &KMax0;

    if ((IMax > 2) &&
        (JMax > 2) &&
        (KMax > 2))
    {
        FieldData_pa V = TecUtilDataValueGetWritableNativeRef(Z,V);
        LgIndex_t I,J,K;
        /*
         * All interior nodes will start at I,J,K = 2 and end
         * at IMax-1,JMax-1,KMax-1
         */
        for (K = 2; K < KMax-1; K++)
            for (J = 2; J < JMax-1; J++)
```

```

        for (I = 2; I < IMax-1; I++)
        {
            LgIndex_t FinalOffset = I + JMax*((J-1) + KMax*(K-1));
            TecUtilDataValueSetByRef(FinalOffset,99.0);
        }
    }
}

```

10 - 1.3 By Specifying Zone-variable

You can access data directly using `TecUtilDataValueSetByZoneVar` and `TecUtilDataValueGetByZoneVar`. These functions allow you to specify the zone, variable and data point you wish to access.

This method is the easiest to use, but it is also the slowest. If your does not need to access large amounts of field data, you may find these functions to be the most convenient as there is no setup required to use these functions.

To set the first data point of the second variable of zone 5 to 3.14, you would call:

```
TecUtilDataValueSetByZoneVar(5,2,1,3.14);
```

To query this value:

```
double Value = TecUtilDataValueGetByZoneVar(5,2,1);
```

When working with this method, please note the following issues:

- All indices are one-based.
- You must be sure that the variable, index, and zone number parameters are valid. If not, this function will issue an error.
- These functions use the double data type, regardless of the field data type of the zone. If the field data type is not double, then these functions will perform an appropriate conversion for you. If needed, you may find out the type of the field data by calling:

```
FieldDataType_e fd_type = TecUtilDataValueGetRefType(FD);
```

10 - 2 Working with Shared Data

Data sharing allows you to lower your use of physical memory. You may share variables and connectivity information between zones. In some cases, data sharing occurs automatically for you. For example, duplicating a zone will share all variables and the connectivity information between the original zone and the newly created zone. Alternatively, you can manually share a variable or connectivity list by calling `TecUtilDataValueShare` or `TecUtilDataConnectShare`, respectively. When sharing a variable, the memory allocated for the variable in the destination zone is freed and the variable in the destination zone will point to the memory used by the variable in the source zone. The share count is then incremented by one. Similarly, for sharing a connectivity list.

10 - 2.1 Branching Shared Data

Some operations will automatically force the branching of shared variables and/or connectivity information. Branching simply means that the variable or connectivity is no longer shared and every zone subsequently has its own copy. For example, if you executed the equation $X = X + 1$ exclusively on zone 1 and prior to the operation zone 1 and zone 2 shared X, the X variable will automatically be branched.

It may be the case that you wish to modify a variable in a zone and also require that these modifications occur exclusively in that zone. If this is the case, and you have no previous knowledge of the sharing of this variable, then it is best to branch the variable prior to modification. If previously shared, branching will allocate memory for the variable (or connectivity information) and make a copy of all the values. The share count for the original data is decremented by 1.

To branch a variable or connectivity list use `TecUtilDataValueBranchShared` or `TecUtilDataConnectBranchShared`, respectively. Branching an already branched variable has no effect.

10 - 2.2 Querying or Modifying Shared Data

Variable and connectivity sharing information may be queried or altered using the standard methods described earlier in this chapter. If the variable or connectivity information is shared, any modifications will be realized by all zones that share the information.

Use one of the following functions to determine if a variable or connectivity list is shared:

<code>TecUtilDataConnectGetShareCount</code>	Get the number of zones sharing connectivity information
<code>TecUtilDataValueGetShareCount</code>	Get the number of zones sharing a variable.
<code>TecUtilConnectGetShareZoneSet</code>	Gets the set of zones that share connectivity with a particular zone.
<code>TecUtilDataValueGetShareZoneSet</code>	Gets the set of zones that share a variable with a particular zone.

State changes can be used to identify the variable (or connectivity information) that was changed. Only those zones that were altered need to be identified, and the Tecplot Engine will recognize that other zones may be affected because of sharing.

10 - 2.3 Allowing Data Sharing

The function `TecUtilDataSetIsSharingAllowed` should be called prior to using any of the `TecUtil` sharing functions, as your customers may shut down sharing. This should be an isolated occurrence and most likely limited to cases where an older add-on must be used that cannot handle shared data and the user is forced to shut down sharing.

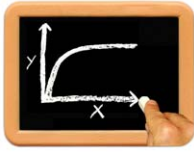
To shut down data sharing, add the following entry to the `tecplot.cfg` file:

```
$!Compatibility AllowDataSharing = No
```

10 - 3 Load-on-demand

When load-on-demand is active, not all of the variables in your data set are necessarily loaded. As such, you should always call one of the `TecUtil` functions listed below before you perform any action that requires a specific variable to be loaded. NOTE: Depending on both the nature of your data set and the nature of your request, the variable you are attempting to load may not be loaded as expected. For example, if you request the nodal values of a given variable (via `TecUtilDataValueGetReadableNLRef`) and only the cell-centered values exist in the data file, the cell-centered values will be loaded and interpolated to nodal values.

<code>TecUtilDataValueGetReadableCCRef</code>	Get a read-only handle to the cell centered data for the specified zone and variable in the data set attached to the current frame.
<code>TecUtilDataValueGetReadableDerivedRef</code>	Get a read-only handle to the derived data for the specified zone and variable in the data set attached to the current frame.
<code>TecUtilDataValueGetReadableNLRef</code>	Get a read-only handle to the node located data for the specified zone and variable in the data set attached to the current frame.
<code>TecUtilDataValueGetReadableNativeRef</code>	Get a read-only handle to the native data for the specified zone and variable in the data set attached to the current frame.
<code>TecUtilDataValueGetWriteableNativeRef</code>	Get a native read/write handle to the data for the specified zone and variable in the data set attached to the current frame.



Example 13: Equate Add-on

The equate add-on is an example of how to query and set field data in an add-on. It will appear in the **Tools** menu as *Equate*. The add-on is used to multiply each data point of the first variable in the first zone by a value entered in a dialog.

The source code shown in this example are included in your Tecplot 360 installation under *adk/samples/equate*.

Step 1 Add-on Setup

Equate uses source code files created by the `CreateNewAddOn` script (Linux/Macintosh). Our project and add-on names will be *Equate*.



Please review [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) and/or [Chapter 3: "Creating Add-ons on Windows Platforms"](#) before proceeding with this tutorial. All of the code from this point on is platform-independent, and you can work through the tutorial using either any environment.

When running `CreateNewAddOn`, answer the questions as follows:

- **Project name (base name)** - Equate
- **Add-on name** - Equate
- **Company name** - [Your Company Name]
- **Type of Add-on** - General Purpose
- **Language** - C
- **Use TGB to create a platform-independent GUI?** - Yes
- **Add a menu callback to the Tecplot "Tools" menu?** - Yes
- **Menu Text** - Equate
- **Menu Callback Option** - Launch a modeless dialog
- **Dialog title** - Equate

After running the `CreateNewAddOn` script, you should have the following files:

```
ADDGLBL.h    guicb.c        guibld.c        guidefs.c
GUIDEFS.h    main.c            gui.lay
```

You will have other files specific to your platform, but only those listed here will be modified. Verify that you can compile your add-on project and load it into Tecplot 360. For UNIX or Linux platforms, this is done by running the `Runmake` script. On Windows platforms, your add-on will appear in the **Tools** menu in Tecplot 360. For detailed information on compiling, refer to [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) or [Chapter 3: "Creating Add-ons on Windows Platforms"](#).

Step 2 Creating the Dialog

Now create your main dialog. This will be displayed when *Equate* is selected from Tecplot 360's **Tools** menu. The dialog will be modeless with a text field, label, and button. When a user enters a numeric value in the text field and selects the button, *Equate* will multiply each data point of the first variable in the first

zone by that value. Before beginning, be sure that Tecplot GUI Builder (TGB) is available from Tecplot 360's **Tools** menu. If TGB is not available, do the following:

On Windows platforms, in the Tecplot 360 home directory edit the file *tecplot.add* and add the line:

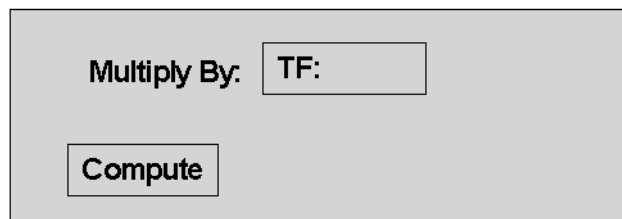
```
$!LoadAddOn "guibld"
```

For Linux or Macintosh platforms, edit the file *tecdev.add* in your Add-on Development Root Directory and add the line:

```
$!LoadAddOn "guibld"
```

To create the main dialog, perform the following steps:

1. Run Tecplot 360 and load the *gui.lay* file for your project. Select Tecplot GUI Builder (TGB) from the Tecplot 360 **Tools** menu.
2. Resize the frame and edit the layout as follows:



You can edit a control by double-clicking on it and editing as you would text.

Although the text fields and buttons are referred to as controls (since they exist in a Tecplot 360 layout file), they are represented as text field objects in Tecplot 360.

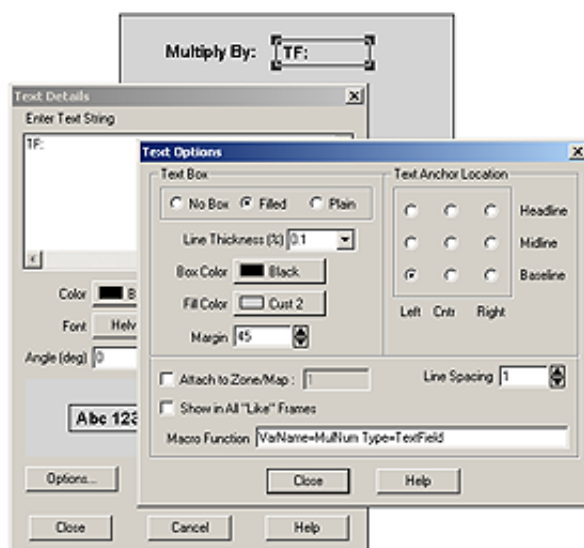
3. So that TGB will create meaningful variable names for the text field controls, change their properties in Tecplot 360. Double-click on the text field "TF;" then select Options.

Do not alter the text string "TF". Tecplot 360 uses this string to identify this control as a Text Field.

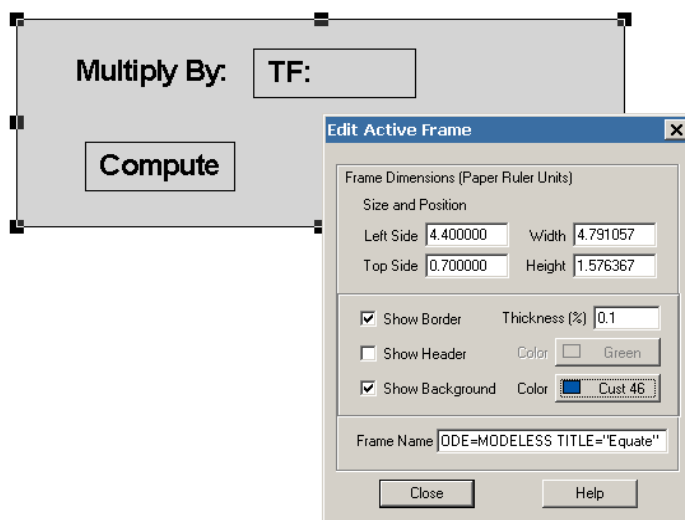
In the Macro Function text field, set VarName=MulNum. This will be the base name of the "Multiply By" callback function, which will be named **MulNum_TF_D1_CB**. TGB takes the base name and decorates it with the dialog number, control type and CB (for callback).

4. Double-click on the [Compute] button, then select Options. Set VarName=Compute in the "Macro Function" field. The [Compute] button callback function will be named **Compute_BTN_D1_CB**. Double-click on the "Multiply By" label, then select Options. Set VarName=MultiplyBy in the "Macro Function" field. Callback functions are not generated for labels, but a variable name will be generated and will be named **MultiplyBy_LBL_D1**.
5. In TGB, the title of the dialog is specified in the **Edit Active Frame** dialog (accessed in the **Frame** menu). Double-click on the dialog frame and verify that the Frame Name has been set to "Equate":

ID=1 MODE=MODELESS TITLE="Equate" OKCLOSEBUTTON=T HELPBUTTON=T



6. You can now build the source for this layout. From the **TGB** dialog, select **Go Build**. If you wish to preview your dialog, select **Preview Layout** from **TGB**.
7. Rename the file *guicb.tmp* to be *guicb.c*, replacing the original *guicb.c* then compile the source code.



Step 3 GUI source code

Now we will examine the source code files generated by TGB.

- Files *guidefs.c*, *GUIDEFS.h*: If you are using C++, be sure that *GUIDEFS.h* is included at the top of the *main.cpp*, and add this code in the *MenuCallback* function:

```
BuildDialog1 (MAINDIALOGID);
TecGUIDialogLaunch (Dialog1Manager);
```

GUIDEFS.h contains the variable names of all of the controls added to the dialog. TGB has taken the variable names specified in the Macro Function Command field and decorated them as follows:

```
int Dialog1Manager      = BADDIALOGID;
```

```
int MulNum_TF_D1      = BADDIALOGID;
int Compute_BTN_D1    = BADDIALOGID;
int MultiplyBy_LBL_D1= BADDIALOGID;
```

TF is text field and Dn is the dialog number. Since there is only one dialog box, n is 1. For example, the name `MulNum_TF_D1` can be decoded as “This variable represents the `MulNum` Text Field in Dialog 1.” The variables are initialized to `BADDIALOGID` to ensure that they cannot be passed as parameters to any `TecGUI` library function until the dialog has actually been created. At that time they will be assigned a valid identification.

Note: Never edit these files directly. TGB will generate them every time you select Go Build.

- File `guibld.c`: Contains the code used to build the dialogs.

Note: Never edit this file directly. TGB will generate this file every time you select Go Build, so any changes you make will be overwritten. Also, this file is never included directly in the project. Instead, the text of this source code file is included directly in `guib.c` with a `#include 'guibld.c'` preprocessor statement at the end of `guib.c`.

- File `guibc.tmp`: Contains all of the callbacks for the dialog controls. A callback function is a function you define which is called by Tecplot 360 when an event occurs for a control. For example, a button control will have a callback function for the button pressed event.

Initially, TGB will generate empty callbacks, but instead of writing them to `guibc.c`, it will write them to a file named `guibc.tmp`. The reason for this is that TGB does not want to overwrite any code that you may have added to `guibc.c`. Thus, whenever you add new controls, you must cut-and-paste the new callback functions in `guibc.tmp` into `guibc.c`. Note that in Step 7 we copied `guibc.tmp` to `guibc.c`. This is what you want to do when you first start the project since at that time there is no custom code in `guibc.c`.

To see the new dialog:

1. Compile the add-on and run Tecplot 360.
2. Select *Equate* from the **Tools** menu.

Step 4 Setting up State Variables and Initializing the Dialog Fields

When the dialog is first displayed, we need to be sure that the `MulNum` text field has a reasonable default value. To avoid using a global variable for `MulNum`, the value will be read from the text field and passed to a function called `Compute`. The text field will then be initialized in the `Dialog1Init_CB` function.

Note the following line in `guibc.c`:

```
/* This is a string because it is put in a dialog text field */
#define DEFAULT_MULNUM "2"
```

Find the following segment of code in `guibc.c` and note the line beginning with `TecGUITextField`.

```
static void Dialog1Init_CB(void)
{
    TecUtilLockStart(AddOnID);

    /*<<< Add init code (if necessary) here>>>*/
    TecGUITextFieldSetString(MulNum_TF_D1,DEFAULT_MULNUM);
    TecUtilLockFinish(AddOnID);
}
```

We have defined the default value to be a string, since that is what `TecGUITextFieldSetString` expects. The `Compute` function will be called when you select `Compute`. The function prototype is follows: `extern void`

Compute (double MulNum); Note the function call to Compute in guicb.c. This function is displayed following. Before calling this function, check that a data set is available. If there is, then it is implied that at least one zone and one variable exist.

Edit guicb.c as follows:

```
static void Compute_BTN_D1_CB(void)
{
    char *strMulNum = NULL;

    TecUtilLockStart(AddOnID);
    strMulNum = TecGUITextFieldGetString(MulNum_TF_D1);

    if (TecUtilDataSetIsAvailable())
    {
        Compute(atof(strMulNum));
    }
    else
        TecUtilDialogErrMsg("No data set available.");

    TecUtilStringDealloc(&strMulNum);

    TecUtilLockFinish(AddOnID);
}
```

No error checking is done on the input string. As an exercise, use TecGUITextFieldGetDouble, TecGUITextFieldSetDouble, and TecGUITextFieldValidateDouble to do error checking for you.

Step 5 Writing the Compute function

The final task is to write the Compute function. This will multiply each data point of the first variable in the first zone by the input parameter, then send a message to Tecplot 360 that the data set has changed. The recommended way for an add-on to get and set field data is with FieldData_pa handles. Examine main.c and note the following function:

```
void Compute(double MulNum)
{
    LgIndex_t IMax;
    LgIndex_t JMax;
    LgIndex_t KMax;
    LgIndex_t i;
    LgIndex_t MaxIndex;
    FieldData_pa FD;
    double Value;
    Set_pa set;

    TecUtilLockStart(AddOnID);

    if (TecUtilZoneIsEnabled(1)          &&
        TecUtilVarIsEnabled(1)          &&
        TecUtilZoneGetType(1) == ZoneType_Ordered &&
        TecUtilDataValueGetLocation(1,1) == ValueLocation_Nodal)
    {
        /* Get the number of data points */
        TecUtilZoneGetInfo(1, /* Zone */
                           &IMax,
                           &JMax,
                           &KMax,
                           NULL, /* XVar */
                           NULL, /* YVar */
                           NULL, /* ZVar */
                           NULL, /* NMap */

```

```

        NULL, /* UVar */
        NULL, /* VVar */
        NULL, /* WVar */
        NULL, /* BVar */
        NULL, /* CVar */
        NULL); /* SVar */

    MaxIndex = IMax * JMax * KMax;

    FD = TecUtilDataValueGetWritableRef(1,1);

    for (i = 1; i <= MaxIndex; i++)
    {
        /* Get the value */
        Value = TecUtilDataValueGetByRef(FD,i);

        /* Change it */
        Value *= MulNum;

        /* And set it back */
        TecUtilDataValueSetByRef(FD,i,Value);
    }

    /* Inform Tecplot that we've changed the data */
    set = TecUtilSetAlloc(FALSE);
    TecUtilSetAddMember(set,1,FALSE); /* Zone 1 */
    TecUtilStateChanged(StateChange_VarsAltered,
        (ArbParam_t)set);
    TecUtilSetDealloc(&set);
}
else
{
    TecUtilDialogErrMsg("This sample add-on only performs an equation on
    "
        "variable 1 of zone 1 and only if the zone is "
        "ordered and the variable is node located.");

    TecUtilLockFinish(AddOnID);
}
}

```

Equate is now complete. Recompile and load it into Tecplot 360. Note that this example add-on is only valid for ordered data as we computed MaxIndex by multiplying the dimensions together.

Step 6 Additional exercises

1. Currently, there is no error checking done on the value entered in the text field. You could enter "ABCDEFGH" and `atof` would convert it into 0.0. This could be fixed by adding error checking to the button callback. Use `TecGUITextFieldValidateDouble` and `TecGUITextFieldGetDouble` for better error checking.
2. Add a multi-selection list box which allows you to select one or more zones from a set.
3. Add a multi-selection list box to select one or more variables from a set.
4. This add-on assumes variable 1 and Zone 1 are "Enabled," which may not be the case. Add error checking to make sure Zone 1 is enabled (`TecUtilZoneIsEnabled`) and variable 1 is enabled (`TecUtilVarIsEnabled`).

Manipulating Data

Plots created with the Tecplot Engine rely on the datasets attached to each frame. You can modify, create, transform, interpolate, duplicate, and delete the data in the current dataset. You can also use the data manipulation capabilities to change the value of some or all of your data parts.

Changes to the dataset do not affect the original data file(s). When you save a layout file, a journal of data operations is saved and those operations are repeated when the layout file is read at a later time. If the data in the file has changed, or the data file is overridden with a different file, the operations are applied to the new data. Alternatively, any datasets that have been modified can be saved to data files. Refer to [Chapter 20: “Implementing Data Journaling”](#) for details on data journaling.

This chapter covers the following sections on data manipulation:

- [Calculating Equations](#)
- [Data Smoothing](#)
- [Coordinate Transformation](#)
- [Zone Creation](#)
- [Data Extraction from an Existing Zone](#)
- [Data Interpolation](#)
- [Irregular Data Point Triangulation](#)

11 - 1 Calculating Equations

You can alter the data in existing zones by using equations. Equations allow you to create new variables or change the values of variables or specific data points. Equations can be specified via the `TecUtilDataAlter` function.

The prototype for `TecUtilDataAlter` is:

```
Boolean_t TecUtilDataAlter ( const char *   Equation,
                             Set_pa       ZoneSet,
                             LgIndex_t    IMin
                             LgIndex_t    IMax
                             LgIndex_t    ISkip
                             LgIndex_t    JMin
```

```

LgIndex_t      JMax
LgIndex_t      JSkip
LgIndex_t      KMin
LgIndex_t      KMax
LgIndex_t      KSkip
FieldDataType_e DestDataType
)

```

Where the parameters are described as:

Equation	Refer to Section 11 - 1.1 "Equation Syntax" for a complete description of how to specify equation syntax.
ZoneSet	Select whether to alter: all zones, all active zones, a range of zones, or no zones.
Min, Max, and Skip parameters	For these parameters, select the index ranges to alter in the selected zones. For Ordered Data, the I-index, J-index, and K-index options correspond to the I, J, and K values in the dataset. For finite-element data, the I-index corresponds to the range of nodes and the J-index corresponds to the cell-centered values. The K-index has no bearing on finite-element data. If you are creating a new variable, the new variable's value is set to zero at any index value that is skipped.
DestDataType	Indicates the data type of the new variable.

11 - 1.1 Equation Syntax

Equations have the following form:

$$LValue = F(RValue1, RValue2, RValue3, \dots)$$

Where: $F()$ - A mathematical expression.

$LValue$ - An existing or new variable.

$RValueN$ - A value (such as a constant, variable value, or index value).

If $LValue$ already exists in the dataset of the active frame, the equation is used to modify that variable. If the variable does not exist, the equation is used to create a new variable as a function of existing variables.

There may be any number of spaces within the equation. However, there cannot be any spaces between the letters of intrinsic-function names or for variables referred to by name.

Equation Variables and Values

A variable is specified in one of the following ways:

- **Its order in the data file** - A variable may be referenced according to its order in the data file, where V1 is the first variable in the data file, V2 is the second, and so forth.

To create a new variable using this specification, you must specify the number of the next available variable (i.e. if there are 5 variables in the dataset), the new variable must be called V6. You will receive an error message, if you attempt to assign an invalid variable number.



You can confirm the number and order of variables in the data file by selecting the [Data Set Info] button in the **Specify Equations** dialog and going to the Zones/Var page of the **Data Set Information** dialog. The variables in the dataset are listed on the right-hand side of the page.

- **By its name** - To reference a variable by its name, enclose the name with curly braces (“{” and “}”). For example, to set V3 equal to the value of the variable named R/RFR, you can enter:

V3 = {R/RFR}

Variable names are not case sensitive. Leading and trailing spaces are also not considered. However, spaces within the variable name are significant.

If two or more variables have the same name, the first variable is used when the variable is referred to by name. So, if both V5 and V9 are named R/rfr, V5 is used.

The curly braces can also be used on the left-hand side of the equation. In this case, if a variable with that name does not exist, a new variable is created with that name for all zones.

- **By a letter code** - Variables and index values may be referenced by the following letter codes:
 - **I** - For I-ordered and finite-element data:

	Finite-element	Ordered
Nodal	I is equal to 1	I is equal to the I-index number
Cell-centered	I is equal to the element number	I is equal to the I-index number

- **J** - For J-ordered and finite-element:

	Finite-element	Ordered
Nodal	J is equal to the node number	J is equal to the J-index number
Cell-centered	J is equal to 1	J is equal to the J-index number

- **K** - For K-ordered and finite-element:

	Finite-element	Ordered
Nodal	K is equal to 1	K is equal to the K-index number
Cell-centered	K is equal to 1	K is equal to the K-index number

- **X** - The variable assigned to the X-axis. In XY-plots, all active mappings must have the same X-variable in order for this variable name to be valid.
- **Y** - The variable assigned to the Y-axis. In XY-plots, all active mappings must have the same Y-variable in order for this variable name to be valid.
- **Z** - The variable assigned to the Z-axis (if in 3D Cartesian).
- **A** - The variable assigned to the Theta-axis for Polar plots. For this variable to be valid, the plot type must be set to Polar Line. In addition, all active mappings must have the same Theta-variable.
- **R** - The variable assigned to the R-axis for Polar plots. The plot type must be Polar Line, and all active mappings must have the same R-variable for this variable name to be valid.
- **U** - The X-component of vectors (if defined).
- **V** - The Y-component of vectors (if defined).
- **W** - The Z-component of vectors (if defined).

- **B** - The value-blanking variable for the first active constraint (if applicable).
- **C** - The contour variable for contour group 1 (if defined in the **Contour Details** dialog).
- **S** - The scatter-sizing variable (if defined in the **Scatter Size/Font** dialog).
- **SOLUTIONTIME** - The current solution time.

Letter codes may be used anywhere on the right-hand side of the equation. Do not enclose them in curly braces.

Those letter codes representing variables (all letter codes except I, J, and K) may be used on the left-hand side of the equation as well.

The variables referenced by the letter codes are for the current frame.

Equation Operators and Functions

Binary Operators

In an equation, the valid binary operators are as follows:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Binary operators have the following precedence:

**	Highest precedence
*, /	
+, -	Lowest precedence

Operators are evaluated from left to right within a precedence level.

Functions

The following functions are available (except where noted, all take a single argument):

SIN	Sine (angle must be specified in radians)
COS	Cosine (angle must be specified in radians)
TAN	Tangent (angle must be specified in radians)
ABS	Absolute value
ASIN	Arcsine (result is given in radians)
ACOS	Arccosine (result is given in radians)
ATAN	Arctangent (result is given in radians)
ATAN2(A,B)	Arctangent of A/B (result is given in radians)
SQRT	Returns the positive square root

LOG, ALOG	Natural logarithm (base e)
LOG10, ALOG10	Logarithm base 10
EXP	Exponentiation (base e); $\text{EXP}(V1)=e^{**}(V1)$
MIN(A,B)	Minimum of A or B
MAX(A,B)	Maximum of A or B
SIGN	Returns -1 if argument is negative, +1 otherwise
ROUND	Round off to the nearest integer
TRUNC	Remove fraction part of a value

Notes

- LOG and ALOG are equivalent functions, as are LOG10 and ALOG10.
- Variables input into trigonometric functions must be in units of radians.
- To call an intrinsic function, place its argument within parentheses, i.e. to set V4 to the arctangent of V1, use:
V4 = ATAN(V1)

Derivative and Difference Functions

The derivative functions can be called in the same manner as described above for intrinsic functions. Derivative and difference functions can be calculated with respect to the following variables:

Variable	Definition	Restricted to
x, y, z	Variable assigned to the x-axis, y-axis or z-axis, respectively	XY Line ^a , 2D or 3D
a	Variable assigned to the theta-axis	Polar Line
r	Variable assigned to the radial-axis	Polar Line
i, j, k	Index range	Ordered Zones

Table 11 - 1: Derivative and Difference Function Variables

- a. If you have multiple X-axes or Y-axes in an XY line plot, the variables assigned to the X and Y-axis in the first available mapping will be used.

The complete set of first and second-derivative and difference functions are listed below:

Type	Function Call	Applicable Variables
First Order	dd■	■= x, y, z, a, or r
Second Order	d2d○2	○= x, y, z, a or r
Second Order (cross derivatives)	d2d▲	▲= xy, yz, xz, az, ar, or rz

The derivative function `ddx` is used to calculate $\frac{\partial}{\partial x}$; `d2dx2` calculates $\frac{\partial^2}{\partial x^2}$; `d2dxy` calculates $\frac{\partial^2}{\partial x \partial y}$.

Type	Function Call	Applicable Variables
First Order	<code>dd</code> ♥	♥ = i, j, or k
Second Order	<code>d2d</code> ☉	☉ = i, j, or k
Second-Order (cross derivatives)	<code>d2d</code> ▼	▼ = ij, jk, or ik

Table 11 - 2: Difference Functions

The difference functions `ddi`, `d2di2`, and so forth, calculate centered differences of their argument with respect to the indices I, J, and K based on the indices of the point. For example:

$$\text{ddi}(V) = \frac{V_{i+1} - V_{i-1}}{2}$$



Difference functions cannot be used for finite element zones.

Boundary Values

Boundary values for first-derivative and difference functions (`ddx`, `ddy`, `ddz`, `ddi`, `ddj`, and `ddk`) of ordered zones are evaluated in one of two methods: simple (default) or complex¹.

For simple boundary conditions, the boundary derivative is determined by the one-sided first derivative at the boundary. This is the same as assuming that the first derivative is constant across the boundary (i.e. the second derivative equal to zero).

For complex boundary conditions, the boundary derivative is extrapolated linearly from the derivatives at neighboring interior points. This is the same as assuming that the second derivative is constant across the boundary (i.e. the first derivative varies linearly across the boundary).

For second-derivatives and differences (`d2dx2`, `d2dy2`, `d2dz2`, `d2dxy`, `d2dyz`, `d2dxz`, `d2di2`, `d2dj2`, `d2dij`, `d2dk2`, `d2djk`, and `d2dik`), these boundary conditions are ignored. The boundary derivative is set equal to the derivative one index in from the boundary. This is the same as assuming that the second derivative is constant across the boundary.



You can create your own derivative boundary conditions by using the index range and the indices options discussed previously.

The use of derivative and difference functions is restricted as follows:

- Derivatives and differences for IJK-ordered zones are calculated for the full 3D volume. The IJK-mode for such zones is not considered.
- If the derivative cannot be defined at every data point in all the selected zones, the operation is not performed for any data point.

1. The `!INTERFACE` parameter in the configuration file `tecplot.cfg` selects the method to use: `!INTERFACE DATA {DERIVATIVEBOUNDARY=SIMPLE}`. Change the parameter SIMPLE to COMPLEX to use the complex boundary condition. Or you can programmatically set this parameter using the `StyleValue` class.

- Derivative functions are calculated using the current frame's axis assignments. Be careful if you have multiple frames with different variable assignments for the same dataset.
- Derivatives at the boundary of two zones may differ since only one zone is operated on at a time while generating derivatives.

Auxiliary Data

You may use auxiliary data containing numerical constants in equations. The syntax for using auxiliary data in equations is:

```
AUXZONE[nnz]:Name
AUXDATASET:Name
AUXFRAME:Name
AUXVAR[nnv]:Name
AUXLINEMAP[nnm]:Name
```

where

nnz = the zone number(s)

nnv = the variable number(s)

nnm = the line map number(s)

Name = name of the auxiliary data

For example, a dataset auxiliary data constant called **Pref** would be referenced using `AUXDataSet:Pref`. Equations using this auxiliary data might appear as:

```
{P} = {P_NonDim} * AUXDataSet:Pref
```

Refer to [Chapter 13: "Auxiliary Data"](#) for more information about auxiliary data and using the `tecplot::toolbox::AuxData` class.

Zone Number Specification

By following a variable reference with square brackets ("[" and "]"), you can specify a particular zone from which to get the variable value.

The zone number must be a positive integer constant less than or equal to the number of zones. The zone specified must have the same structure (I, IJ, or IJK-ordered or finite-element) and dimensions (**IMax**, number of nodes) as the zone(s) the equation(s) will be applied to.



If you do not specify a zone, the zone modified by the left-hand side of the equation is used.

Zone specification works only on the right-hand side of the equation.

Index Specification

By following a variable reference with parentheses, you can specify indices for ordered data only. Indices can be absolute or an offset from the current index.

Index offsets are specified by using the appropriate index "i", "j" or "k" followed by a "+" or "-" and then an integer constant. Any integer offsets may be used. If the offset moves beyond the end of the zone, the boundary value is used. For example, `V3(i+2)` uses the value `V3(IMAX)` when $I=IMax-1$ and $I=IMax$. `V3(I-2)` uses the value of `V3(1)` when $I=1$ or $I=2$.

Absolute indices are specified by using a positive integer constant only. For example, V3(2) references V3 at index 2, regardless of the current i index.

If the indices are not specified, the current index values are used.

Variable Sharing between Zones

For zones with the same structure and index ranges, you can set a variable to be shared by specifying that the variable for those zones have the values from one zone. For example, if zones 3 and 4 have the same structure and you compute V3=V3[3] for zones 3 and 4, V3 will be shared.



Subsequent alteration of the variables may result in loss of sharing. Refer to [Chapter 11: "Accessing Data"](#) for more information on data sharing.

11 - 2 Data Smoothing

You can smooth the values of a variable of any zone (in either 2D or 3D) to reduce “noise” and lessen discontinuities in data. Smoothing can also be used after inverse-distance interpolation to reduce the artificial peaks and plateaus. Each pass of smoothing shifts the value of a variable at a data point towards an average of the values at its neighboring data points. Data smoothing can be performed with the TecUtilSmooth function.

The prototype for TecUtilSmooth is:

```
Boolean_t TecUtilSmooth (   EntIndex_t      Zone,
                           EntIndex_t      SmoothVar,
                           LgIndex_t       NumSmoothPasses
                           double          SmoothWeight
                           BoundaryCondition_e SmoothBndryCond
                           )
```

Where the parameters are described as:

Zone	Specify the number of the zone to smooth. The zone must be an ordered zone (not a finite-element zone).
SmoothVar	Select the number of the variable to smooth. For the XY Line plot type, the variable must be a dependent variable for one active mapping for that zone.
NumSmoothPasses	Specify the number of smoothing passes to perform. The default is one. A greater number of passes results in greater smoothing, but takes more time.
SmoothWeight	Specify the relaxation factor for each pass of smoothing. Enter a number between zero and one (exclusively). Large numbers flatten peaks and noise quickly. Small numbers smooth less each pass, rounding out peaks and valleys rather than eliminating them. The normal default is 0.8.
SmoothBndryCond	Select the boundary conditions by which to smooth.

11 - 2.1 Limitations to Smoothing

- Finite-element zones cannot be smoothed with anything other than fixed boundary conditions.
- The current frame’s axis assignments are used to determine the variables to use for the coordinates in the smoothing and also to determine whether the smoothing should be done

with XY Line, 2D, or 3D Cartesian plot types. Be careful if you have multiple frames with different variable assignments for the same dataset.

- Any axis scaling is ignored while smoothing.
- For I-ordered or finite-element line segment zones, the current frame can be in the XY Line, 2D or 3D Cartesian plot types. In XY Line, the variable must be the dependent variable of one active mapping for that zone.
- For IJ-ordered, finite-element triangle, or finite-element quadrilateral zones, the current frame can be a 2D or 3D Cartesian plot type, but you cannot smooth the variables assigned to the X and Y-axes in 2D Cartesian.
- For IJK-ordered, finite-element tetrahedral, or finite-element brick zones, the plot type must be 3D Cartesian, and you cannot smooth the variables assigned to the X, Y, and Z-axes. The IJK-mode is ignored. The zone is smoothed with respect to the entire 3D volume.
- Smoothing does not extend across zone boundaries. If you use a boundary condition option other than Fixed (such that values along the zone boundary change), contour lines can be discontinuous at the zone boundaries.
- Smoothing is performed on all nodes of a zone, and disregards value-blanking.

11 - 3 Coordinate Transformation

By default, all 2D and 3D plots use a Cartesian coordinate system with X, Y, and Z-axes. If your data is in polar coordinates or spherical coordinates, you will probably want to compute the corresponding Cartesian (X,Y and Z) coordinates before visualizing your data. Coordinate transformation can be performed with the `TecUtilTransformCoordinatesX` function.

The prototype for `TecUtilTransformCoordinatesX` is:

```
Boolean_t TecUtilTransformCoordinatesX ( ArgList_pa ArgList )
```

Argument list descriptions are:

SV_CREATENEWVARIABLES	Specify whether to create new variables for the transformed coordinates, or place them in specified variable numbers.
Source Variables	Specify the source variables for each coordinate. If new variables are not created, these are required destination variable numbers for coordinate transformations.
SV_TRANSFORMATION	<p>Select the type of transformation for changing all points in one or more zones from one coordinate system to another.</p> <p>Polar to Rectangular - The current Y-variable represents the radius r, and the current X-variable represents the angle θ.</p> <p>Spherical to Rectangular - The current Y-variable represents the radius r, the current X-variable the angle θ (in radians), and the current Z-variable the angle ψ.</p>

Refer to [Chapter 23: "Argument Lists"](#) for more information on argument lists.

11 - 4 Zone Creation

You can create a zone programmatically using any of the following techniques:

- [1D Line Creation](#)
- [Rectangular Zone Creation](#)

- [Circular or Cylindrical Zone Creation](#)
- [Zone Duplication](#)
- [Mirror Zone Creation](#)
- [FE Surface Zone Creation \(from Polylines\)](#)
- [Zone Creation by Entering Values](#)
- [Sub-zone Extraction](#)



When you create a zone from calculated (i.e. not loaded) data, you are forced to load all of the zones (regardless of your load-on-demand settings).

11 - 4.1 1D Line Creation

A 1D-line zone is an I-ordered set of points along a line. To create a 1D-line zone, call the `TecUtilCreateSimpleZone` function. The prototype for `TecUtilCreateSimpleZone` is:

```
Boolean_t TecUtilCreateSimpleZone (  LgIndex_t      NumPoints,
                                   const double*      V1Values,
                                   const double*      V2Values,
                                   FieldDataType_e      FieldDataType
                                   )
```

Where the parameters are described as:

NumPoints	Enter the number number of XY pairs of data.
V1Values	Enter an array of X (or theta) values for the zone.
V2Values	Enter an array of Y (or R) values for the zone.
FieldDataType	Specify the data type for the variables in the new zone.

The points are uniformly distributed along the X-axis between **XMin** and **XMax**. Y, and any other variables, are set to zero.



You can create a 1D line zone as the first step in plotting an analytic function and then specify an equation that modifies the Y-variable of the new zone using `TecUtilDataAlter`.

11 - 4.2 Rectangular Zone Creation

You can create a new ordered rectangular zone with the dimensions in the I-, J- and K-directions you specify. The zone that you create has the same number of variables as other zones in the dataset. To create a rectangular zone, call the `TecUtilCreateRectangularZone` function.

The prototype for `TecUtilCreateRectangularZone` is:

```
Boolean_t TecUtilCreateRectangularZone (  LgIndex_t      IMax,
                                           LgIndex_t      JMax,
                                           LgIndex_t      KMax,
                                           double          XMin
                                           double          YMin
                                           double          ZMin
                                           double          XMax
                                           double          YMax
```

```

double      ZMax
FieldDataType_e  FieldDataType
)

```

Where the parameters are described as:

Dimensions - Enter the number of data points in the I, J and K-directions.	
IMax	I-Dimension of the zone to create.
JMax	J-Dimension of the zone to create.
KMax	K-Dimension of the zone to create.
Coordinates - Enter the start and end points of the physical coordinates (X,Y and Z).	
XMin	X min (occurs at I = 1) for the rectangular zone.
YMin	Y min (occurs at J = 1) for the rectangular zone.
ZMin	Z min (occurs at K = 1) for the rectangular zone.
XMax	X max (occurs at I = I-Max) for the rectangular zone.
YMax	Y max (occurs at J = J-Max) for the rectangular zone.
ZMax	Z max (occurs at K = K-Max) for the rectangular zone.
FieldDataType	Specify the data type for the variables in the new zone.

- To create an I-ordered zone, enter one for both the J- and K-dimensions.
- To create an IJ-ordered zone, enter one for the K-dimension. The z-axis variable will equal ZMin throughout the created zone.
- To create an IJK-ordered zone, enter a K-dimension greater than one.

The data points will be uniformly distributed in the I, J and K directions. Any variable not assigned to an axis is set to zero. You can modify the X, Y, and Z coordinates, and the values of the other variables as well.

11 - 4.3 Circular or Cylindrical Zone Creation

You can create an ordered circular or cylindrical zone with the dimensions in the I, J, and K directions you specify. The I-dimension determines the number of points on each radius of the zones. The J-dimension determines the number of points around the circumference. The K-dimension determines the number of layers in the zone, creating a cylinder.

The zone that you create has the same number of variables as other zones in the dataset. If you have no current dataset, one with two or three variables is created, depending on the K-dimension. If you specify $K=1$, the dataset is created as IJ-ordered, and has two variables. If you specify $K>1$, the dataset is created as IJK-ordered, and has three variables. To create a circular zone, call the `TecUtilCreateCircularZone` function.

The prototype for `TecUtilCreateCircularZone` is:

```

Boolean_t TecUtilCreateCircularZone (  LgIndex_t      IMax,
                                       LgIndex_t      JMax,
                                       LgIndex_t      KMax,
                                       double          XOrigin,
                                       double          YOrigin,
                                       double          Radius,

```

```

double      ZMin,
double      ZMax,
FieldDataType_e  FieldDataType
)

```

Where the parameters are described as:

IMax	Select the number of points in the radial direction (I).
JMax	Select the number of points in the circumferential direction (J).
KMax	Select the number of points for the height of the cylinder (K). Set K equal to one to create a 2D circular zone.
Coordinates - Enter the start and end points of the physical coordinates (X,Y and Z).	
XOrigin	Specify the coordinates for the X-origin of the zone center.
YOrigin	Specify the coordinates for the Y-origin of the zone center.
Radius	Set the length of the radius.
ZMin	Set the minimum Z-coordinates.
ZMax	Set the maximum Z-coordinates.
FieldDataType	Specify the data type for the variables in the new zone.

For 2D (IJ-ordered), a zone in which I-circles are connected by J-radial lines is created, as shown in [Figure 11-1 \(A\)](#). For 3D ($K>1$), a K-layered cylindrical zone having I-circles connected by J-radial planes is created, as shown in [Figure 11-1 \(B\)](#). All other variables are set to zero.

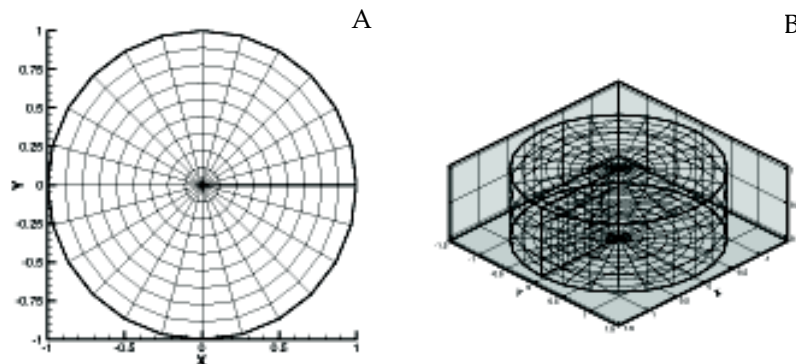


Figure 11-1. (A) A 2D circular zone (B) A 3D circular zone.

11 - 4.4 Zone Duplication

You can create a full duplicate of one or more existing zones. Each duplicate zone has the same name as its source zone. To duplicate a zone or zones, call the `TecUtilZoneCopy` function.



After a zone is duplicated, all variables in the newly created zone(s) will be shared with their corresponding source zone(s).

The prototype for TecUtilZoneCopy is:

```
Boolean_t TecUtilZoneCopy (  EntIndex_t  ZoneUsed,
                             LgIndex_t   IMin,
                             LgIndex_t   IMax,
                             LgIndex_t   ISkip,
                             LgIndex_t   JMin,
                             LgIndex_t   JMax,
                             LgIndex_t   JSkip,
                             LgIndex_t   KMin,
                             LgIndex_t   KMax,
                             LgIndex_t   KSkip
                           )
```

Where the parameters are described as:

ZoneUsed	Specify the source zone. Must be greater than or equal to one.
IMin	Specify the minimum I-index.
IMax	Specify the maximum I-index.
ISkip	Specify the I skip value. Set to one to duplicate the entire I-direction.
JMin	Specify the minimum J-index.
JMax	Specify the maximum J-index.
JSkip	J skip value. Set to one to duplicate the entire J-direction.
KMin	Specify the minimum K-index.
KMax	Specify the maximum K-index.
KSkip	K skip value. Set to one to duplicate the entire K-direction.

See also [Section 11 - 5.1 "Sub-zone Extraction"](#) on page 101.

11 - 4.5 Mirror Zone Creation

You can also create a duplicate zone that is the mirror image of an existing zone. To create mirrored zones, call the TecUtilCreateMirrorZones function.



You can only create mirrored zones along one of the standard axes (2D) or the plane determined by any two axes (3D).

The prototype for TecUtilCreateMirrorZones is:

```
Boolean_t TecUtilCreateMirrorZones (  Set_pa  SourceZones,
                                     char     MirrorVar,
                                   )
```

Where the parameters are described as:

SourceZones	Specify the set of zones to mirror.
MirrorVar	Specify the axis (2D) or axis plane (3D) to mirror about.

Each mirror zone has a name of the form “Mirror of zone *sourcezone*”, where *sourcezone* is the number of the zone from which the mirrored zone was created.



The variables in the newly created zone(s) are shared with their corresponding source zone(s), except for the coordinate and velocity normal to the symmetry plane.

11 - 4.6 FE Surface Zone Creation (from Polylines)

You can create a finite-element surface zone from two or more I-ordered zones. To create FE surface zones from polylines, call the `TecUtilGeom2DPolylineCreate` function.



The data must be arranged in non-intersecting polylines, where each polyline can have any number of points.

The prototype for `TecUtilGeom2DPolylineCreate` is:

```
Boolean_t TecUtilGeom2DPolylineCreate ( CoordSys_e PositionCoordSys,
                                         double *   PtsX_Array,
                                         double *   PtsY_Array,
                                         LgIndex_t   NumPts
                                         )
```

Where the parameters are described as:

PositionCoordSys	Select the coordinate system to use to position the geometry.
PtsX_Array	Specify the array of X-coordinates of the polyline.
PtsY_Array	Specify the array of Y-coordinates of the polyline.
NumPts	Select two or more zones to create your new zone from. The field displays only I-Ordered zones (the polylines).

Create Zone from Polylines is useful when:

- Data is collected on the surface of an irregularly shaped object.
- Measurements were taken at various depths and distances within a fluid.

11 - 4.7 Zone Creation by Entering Values

You can create an I-ordered zone for XY-plots from manually entered values using the `TecUtilCreateSimpleZone` function. See [Section 11 - 4.1 “1D Line Creation”](#) for details.

11 - 5 Data Extraction from an Existing Zone

You may create new zones by extracting (or interpolating) data from existing zones in a number of ways. Derived objects, such as contour lines, FE-boundaries, iso-surfaces, slices, or streamtraces may be extracted to be independent zones. You may also extract data using a specified slice plane, discrete points, points from a polyline, or points from a geometry.

11 - 5.1 Sub-zone Extraction

You can use `TecUtilZoneCopy` to create a sub-zone of an existing zone.



Sub-zone Extraction is available for ordered zones only.

A sub-zone can be created by using any of the following options:

- Use `TecUtilCreateSliceZoneFromPlane` to extract data using a specified slice plane.
- Use `TecUtilExtractInstallCallback` to extract data using discrete points.
- Use `TecUtilExtractFromPolyline` to extract data using points from a polyline.
- Use `TecUtilExtractFromGeom` to extract data using points from a geometry.

See [Section 11 - 4.4 "Zone Duplication"](#) for more information on the `TecUtilZoneCopy` function.

11 - 6 Data Interpolation

Interpolation refers to assigning new values for the variables at data points in a zone based on the data point values in another zone (or set of zones).

For example, you may have a set of data points in an I-ordered zone that are distributed randomly in the XY-plane. This type of data is sometimes referred to as unordered, ungridded, or random data. In Tecplot 360, it is referred to as irregular data. Using data in this form, you can create mesh plots and scatter plots, but you cannot create contour plots, light-source shading, or streamtraces.

You can interpolate the irregular I-ordered data onto an IJ-ordered mesh, and then create contour plots and other types of field plots with the interpolated data. You can also interpolate your 3D, I-ordered irregular data into an IJK-ordered zone and create 3D volume plots from the IJK-ordered zone. You can even interpolate to a finite-element zone.

The accuracy of the interpolation will depend on your data, the density of the destination grid, how well the grid fits the area of your unorganized zone and the settings used for interpolation.

There are three types of interpolation available:

- [Linear Interpolation](#) - Interpolate using linear interpolation from a set of finite-element, IJ-ordered, or IJK-ordered zones to one zone.
- [Inverse-distance Interpolation](#) - Interpolate using an inverse-distance weighting from a set of zones to one zone.
- [Kriging Interpolation](#) - Interpolate using kriging interpolation from a set of zones to one zone.

11 - 6.1 Linear Interpolation

Linear interpolation interpolates data from one or more ordered or finite-element zones onto a destination zone. Irregular I-ordered data cannot be used for the source zones in linear interpolation. (For 2D data, you may be able to first create a finite-element zone from an irregular, I-ordered zone by using triangulation.)

Linear interpolation finds the values in the destination zone based on their location within the cells of the source zones. The value is linearly interpolated to the destination data points using only the data points at the vertices of the cell (or element) in the source zone(s).

The prototype for `TecUtilLinearInterpolate` is:

```
Boolean_t TecUtilLinearInterpolate ( Set_pa      SourceZones,
                                     EntIndex_t   DestZone,
```

```

        Set_pa          VarList
        double          LinearInterpConst
        LinearInterpMode_e LinearInterpMode
    )

```

Parameter descriptions are:

SourceZones	Specify the set of zones used to obtain the field values for interpolation.
DestZone	Specify the destination zone for interpolation.
VarList	Specify the set of variables to interpolate.
LinearInterpConst	Constant value to which all points outside the data field are set.
LinearInterpMode	<p>Select how to treat points that lie outside the source-zone data field.</p> <p>You have two options:</p> <p>LinearInterpMode_SetToConst - Set all points outside the data field to a constant value that you specify.</p> <p>LinearInterpMode_DontChange - Preserve the values of points outside the data field. This is appropriate in cases where you are using one interpolation algorithm inside the data field, and another outside.</p>

11 - 6.2 Inverse-distance Interpolation

Inverse-distance interpolation averages the values at the data points from one set of zones (the source zones) to the data points in another zone (the destination zone). The average is weighted by a function of the distance between each source data point to the destination data point. The closer a source data point is to the destination data point, the greater its value is weighted.

In many cases, the source zone is an irregular dataset—an I-ordered set of data points without any mesh structure (a list of points). Inverse-distance interpolation may be used to create 2D or 3D surfaces, or 3D volume field plots of irregular data. The destination zone can, for example, be a circular or rectangular zone created within the Tecplot Engine. To perform inverse-distance interpolation, call the `TecUtilInverseDistInterpolation` function.

The prototype for `TecUtilInverseDistInterpolation` is:

```

Boolean_t TecUtilInverseDistInterpolation (Set_pa      SourceZones,
                                           EntIndex_t  DestZone,
                                           Set_pa      VarList
                                           double      InvDistExponent
                                           double      InvDistMinRadius
                                           PtSelection_e InterpPtSelection
                                           LgIndex_t   InterpNPoints
                                           )

```

Where the parameters are described as:

SourceZones	Specify the zones to be interpolated
DestZone	Specify a zone into which to interpolate. Existing values in the destination zone will be overwritten.
VarList	Specify which variables are to be interpolated.
InvDistExponent	Specify the exponent for the inverse-distance weighting.

InvDistMinRadius	Specify the minimum distance used for the inverse-distance weighting. Source data points which are closer to a destination data point than this minimum distance are weighted as if they were at the minimum distance. This tends to reduce the peaking and plateauing of the interpolated data near the source data points.
InterpPtSelection	<p>Specify the method used for determining which source points to consider for each destination point.</p> <p>There are three available methods, as follows:</p> <p>Nearest N - For each point in the destination zone, consider only the closest n points to the destination point. These n points can come from any of the source zones. This option may speed up processing if n is significantly smaller than the entire number of source points.</p> <p>Octant - Like Nearest N above, except the n points are selected by coordinate-system octants. The n points are selected so they are distributed as evenly as possible throughout the eight octants. This reduces the chances of using source points which are all on one side of the destination point.</p> <p>All - Consider all points in the source zone(s) for each point in the destination zone.</p>
InterpNPoints	Specify the number of source points to consider for each destination data point.



The exponent should be set between 2 and 5. The algorithm is speed-optimized for an exponent of 4, although in many cases, the interpolation looks better with an exponent of 3.5.

Inverse-distance interpolation ignores the IJK-mode of IJK-ordered zones. All data points in both the source and destination zones are used in the interpolation.



The current frame's axis assignments are used to determine the variables to use for coordinates in interpolation. However, axis scaling is ignored.

Inverse-Distance Algorithm

The algorithm used for inverse-distance interpolation is simple. The value of a variable at a data point in the destination zone is calculated as a function of the selected data points in the source zone.

The value at each source zone data point is weighted by the inverse of the distance between the source data point and the destination data point (raised to a power) as shown below:

$$\varphi_d = \frac{\sum w_s \varphi_s}{\sum w_s} \quad (\text{summed over the selected points in the source zone})$$

where φ_d and φ_s are the values of the variables at the destination point and the source point, respectively, and w_s is the weighting function defined as:

$$w_s = D^{-E}$$

D in the equation above is the distance between the source point and the destination point or the minimum distance, whichever is greater.

Smoothing may improve the data created by inverse-distance interpolation. Smoothing adjusts the values at data points toward the average of the values at neighboring data points, removing peaks, plateaus, and noise from the data.



Kriging and Inverse Distance Interpolation Improvements: For better results with 3D data, try changing the range of your Z-variable to one similar to the X-range the Y-range. Also, set Zero Value to 0.05.

11 - 6.3 Kriging Interpolation

Kriging is a more complex form of interpolation than inverse-distance. It generally produces superior results to the inverse-distance algorithm but requires more computer memory and time.

The prototype for TecUtilKrig is:

```
Boolean_t TecUtilKrig (  Set_pa      SourceZones,
                        EntIndex_t  DestZone,
                        Set_pa      VarList
                        double       KrigRange
                        double       KrigZeroValue
                        Drift_e      KrigDrift
                        PtSelection_e InterpPtSelection
                        LgIndex_t    InterpNPoints
                        )
```

Where the parameters are described as:

SourceZones	Specify the zones to be interpolated.
DestZone	Specify a zone into which to interpolate. Existing values in the destination zone will be overwritten.
VarList	Specify which variables are to be interpolated.
KrigRange	Select the distance beyond which source points become insignificant for the kriging. The value is stated as the fraction of the length of the diagonal of the box which contains the data points. A range of zero means that any point not coincident with the destination point is statistically insignificant. A range of one means that every point in the dataset is statistically significant for each point. In general, values between 0.2 and 0.5 should be used.
KrigZeroValue	Select the semi-variance at each source data point on a normalized scale from zero to one. Semi-variance is the certainty of the value at a data point. Zero is usually a good number for the zero value, and it causes the interpolated data to fit closely to all the source data points. Values greater than zero mean the values at the source points have some uncertainty or noise. Increasing the zero value results in smoother interpolated values that fit increasingly more to the average of the source data.
KrigDrift	Select the overall trend for the data.

InterpPtSelection	<p>Select the method used for determining which source points to consider for each destination point.</p> <p>There are three available methods, as follows:</p> <p>Nearest N - For each point in the destination zone, consider only the closest n points to the destination point. These n points can come from any of the source zones.</p> <p>Octant - Like Nearest N above, except the n points are selected by coordinate-system octants. The n points are selected so they are distributed as evenly as possible throughout the eight octants. This reduces the chances of using source points which are all on one side of the destination point.</p> <p>All - Consider all points in the source zone(s) for each point in the destination zone. In general, you should not use the All option unless you have very few source points.</p>
InterpNPoints	Specify the number of source points to consider for each destination data point.



SourceZones - The current frame's axis assignments are used to determine the variables to use for coordinates in kriging. However, any axis scaling is ignored.

KrigDrift - If the **Drift** is set to Linear or Quadratic, the points selected must be non-collinear (non-coplanar in 3D). To avoid this limitation, set the **Drift** to None.

Alternatively, you can eliminate coincident points by [Irregular Data Point Triangulation](#) before you interpolate.

InterpPtSelection - This option is important for kriging, because kriging involves the computationally expensive inversion and multiplication of matrices. The computational time and memory requirements increase rapidly as the number of selected source data points increases.

Kriging Algorithm

For a detailed discussion of the kriging algorithm refer to: Davis, J. C., *Statistics and Data Analysis in Geology*, Second Edition, John Wiley & Sons, New York, 1973, 1986.



Kriging and Inverse Distance Interpolation Improvements: For better results with 3D data, try changing the range of your Z-variable to one similar to the X-range the Y-range. Also, set Zero Value to 0.05.

11 - 7 Irregular Data Point Triangulation

Triangulation is a process that connects data points to form triangles. You can use triangulation to convert irregular, I-ordered datasets into a finite-element surface zone. Triangulation is one of the two options for creating 2D field plots from irregular data. The other is interpolation. Triangulation preserves the accuracy of the data by creating a finite-element surface zone with the source data points as nodes and a set of triangle elements.

Triangulation works best for 2D data. However, you can triangulate 3D surface data, provided the Z-coordinate is single-valued (the surface does not wrap around on itself). When you triangulate 3D surface data, the Z-coordinate of the data is ignored, causing a less-than-optimal triangulation in some cases. To perform triangulation, call the `TecUtilTriangulate` function.

The prototype for `TecUtilTriangulate` is:

```
Boolean_t TecUtilTriangulate ( Set_pa    SourceZones,
                              Boolean_t  DoBoundary,
                              Set_pa    BoundaryZones
                              Boolean_t  IncludeBoundaryPts)
```

```

LgIndex_t * NumCoincidentPts
double      TriangleKeepFactor
)

```

Where the parameters are described as:

SourceZones	Specify the zones to triangulate.
DoBoundary	If TRUE, BoundaryZones must specify one or more I-ordered zones that define the boundaries across which no triangles can be created.
BoundaryZones	Specify a boundary zone for the triangulation. The boundary zones define the boundaries in the triangulation region. If you do not include boundary zones, the data points are assumed to lie within a convex polygon and that all points in the interior can be connected.
IncludeBoundaryPts	Set this to TRUE if you also want to include the points in the boundary zones in the triangulated zone.
NumCoincidentPts	Returns the number of coincident points.
TriangleKeepFactor	This factor is used to define "bad" triangles on the outside of the triangulated zone. At the completion of triangulation, the Tecplot Engine attempts to remove bad triangles from the outside of the triangulation. The definition of a bad triangle is stored as a number between zero (three colinear points) and 1.0 (an equilateral triangle). Typical settings are values between 0.1 and 0.3; settings above 0.5 are not allowed. Bad triangles will not be removed if removing the triangle strands a data point.

After triangulating your data, you can use the resulting finite-element surface zone to create plots. Generally, you turn off the original zone(s) and plot the new zone only, but you can, for example, plot a scatter plot of the original zone(s) along with the contours of the new zone.

Auxiliary Data

12 - 1 Understanding Auxiliary Data

Auxiliary data is the metadata assigned to a variable, linemap, zone, dataset, frame or page. Auxiliary data can be used for many purposes, such as annotating your plot with dynamic information (refer to [Section “Dynamic Text”](#) on page 249 for details).

In the Tecplot Engine, auxiliary data is a name-value pair. The name must begin with an underscore or letter, and may be followed by one or more of the following: underscore, period, letter, or digit characters. An auxiliary data name may not contain spaces and must be a string.

Each auxiliary data location maintains its own set of auxiliary data. This way you may have separate auxiliary data for each zone within a data set, or each frame within a page, etc. Note that you should be careful to use unique names within an auxiliary data location.

Managing auxiliary data is fairly easy when using the `tecplot::toolbox::AuxData` class provided by the Tecplot Toolbox. Note that the `AuxData` class does not actually maintain the auxiliary data, it simply provides an interface with which to interact with stored auxiliary data stored in the Tecplot Engine. To use the Tecplot Toolbox, you must include `tptoolbox.h` in your project.

The Tecplot Engine supports only string values for auxiliary data. However, often you may want to assign other data types to auxiliary data. The Tecplot Toolbox `AuxData` class makes setting and getting auxiliary data of other types easy. Data types currently supported are `char*`, `LgIndex_t`, `double`, `Boolean_t`, and `tecplot::toolbox::Set`. *It is important to note that since all auxiliary data are treated as strings, some precision may be lost when using the `AuxData` class.*



Example 14: Adding Auxiliary Data to a Data Set

```
AuxDataLocation_e auxDataLocation = AuxDataLocation_DataSet;  
const char *auxDataName = "NumBanannas";  
const int numBanannas = 5;  
AuxData dataSetAuxData(auxDataLocation);  
dataSetAuxData.set(auxDataName, numBanannas);
```

This object, as with `TecUtil` functions, operates on the current frame in the current page. You may create an `AuxData` object and reuse it in any frame on any page provided that the current frame has the data required for the `AuxData` object. In the example above, the current frame must have a dataset attached in order for the set method to succeed. The method `AuxData::isValid` is provided to confirm that the state of the current frame is valid for the given `AuxData` object.

Refer to the [ADK Reference Manual](#) for a complete listing of the `AuxData` class methods.

12 - 2 Using Standardized Auxiliary Data

You can use auxiliary data, or meta-data, for many uses. Use it to link to a variable, linemap, zone, dataset, frame or page. Access auxiliary data with the **Data Set Information** dialog in Tecplot 360.

The following table contains a partial list of auxiliary variable names used by the Tecplot Engine. Most of these names pertain to fluid dynamics or related physical processes. Data loaders will generally make assignments to these names for subsequent use by some type of post processing add-on. Note that add-ons store all auxiliary data in Tecplot 360 as character strings. The Type column below indicates the type of data these character strings represent.

Name	Type	Auxiliary data is assigned to
Common.Incompressible	Boolean	Data set
Common.Density	double	Data set
Common.SpecificHeat	double	Data set
Common.SpecificHeatVar	int	Data set
Common.GasConstant	double	Data set
Common.GasConstantVar	int	Data set
Common.Gamma	double	Data set
Common.GammaVar	int	Data set
Common.Viscosity	double	Data set
Common.ViscosityVar	int	Data set

Common.Conductivity	double	Data set
Common.ConductivityVar	int	Data set
Common.AngleOfAttack	double	Data set
Common.SpeedOfSound	double	Data set
Common.ReferenceU	double	Data set
Common.ReferenceV	double	Data set
Common.UVar	int	Data set
Common.VVar	int	Data set
Common.WVar	int	Data set
Common.VectorVarsAreVelocity	Boolean	Data set
Common.PressureVar	int	Data set
Common.TemperatureVar	int	Data set
Common.DensityVar	int	Data set
Common.StagnationEnergyVar	int	Data set
Common.MachNumberVar	int	Data set
Common.Axisymmetric	Boolean	Data set
Common.AxisOfSymmetryVarAssignment ^a	int	Data set
Common.AxisValue	double	Data set
Common.SteadyState	Boolean	Data set
Common.TurbulentKineticEnergyVar	int	Data set
Common.TurbulentDissipationRateVar	int	Data set
Common.TurbulentViscosityVar	int	Data set
Common.TurbulentDynamicViscosityVar	int	Data set
Common.TurbulentFrequencyVar	int	Data set
Common.Gravity	double	Data set
Common.IsBoundaryZone	Boolean	Zone
Common.BoundaryCondition ^b	BCondition	Zone
Common.Time	double	Zone
Common.Mean	double	Variable
Common.Median	double	Variable
Common.Variance	double	Variable
Common.StdDev	double	Variable
Common.AvgDev	double	Variable

Common.GeoMean	double	Variable
Common.ChiSqre	double	Variable

- a. Boolean can be assigned to any of "ON," "OFF," "TRUE," "FALSE," "YES," "NO," "1", "0", "YEP", "NOPE".
- b. Boundary conditions include "Inflow," "Outflow," "Wall," "Slip Wall," "Symmetry," and "Extrapolated."

The use of auxiliary data is optional.

Setting Plot Style

The Tecplot Engine provides a large number of plotting capabilities, including a full range of Sketch¹, XY, Polar, 2D, and 3D plots. While you are likely familiar with the fundamental concepts of data visualization, it is important that you understand the nomenclature and concepts behind data visualization as it pertains to the Tecplot Engine. For background information on any of the plot options described in this chapter, please refer to the [User's Manual](#) (included in your installation).

Once data has been loaded and associated with a particular plot type, the next step is typically to customize the style of the plot. The “style” of the plot encompasses the active plot layers and their settings (such as color, line type, etc.), along with the axis settings, and many display settings. This chapter outlines many of those style settings available to you, along with example code showing how to set these styles.



Although the example code in this chapter provides a good method to connect the desired style settings with actual code, we recommend working with the Code Generator described in [Chapter 16: “Code Generator”](#).

13 - 1 Linemaps and Fieldmaps

Once a dataset has been loaded, the next step is to apply style to create a plot. Setting the style for some objects (e.g. legends, axes, or other annotations) is straightforward because they do not have a direct

1. A Sketch plot is a plot that contains text, geometries, and/or images, but no dataset.

connection to a dataset. However, setting the style for objects that represent the data involves the use of a linemap (for [Line Plots](#)) or a fieldmap (for [Field Plots](#)).

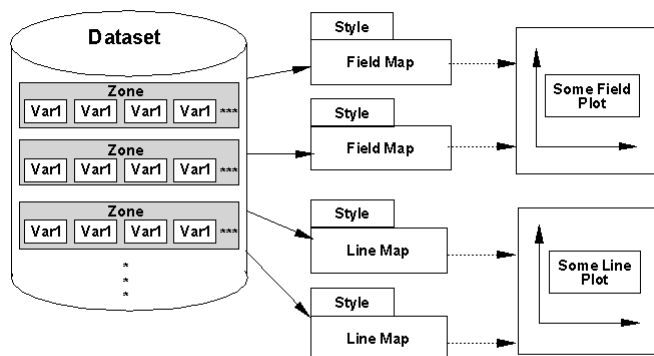


Figure 13-1. Connection of data and style to your plot.

[Figure 13-1](#) depicts the relationship between your dataset, fieldmaps, linemaps and your final plot. The data in a given dataset can map to one or more fieldmaps or linemaps. Each fieldmap and linemap has its own style settings. A field plot is composed of one or more fieldmaps, and a line plot is composed of one or more linemaps.

The following table outlines the information that is stored in fieldmaps and linemaps with respect to a dataset:

Fieldmaps	Set of Zones
Linemaps	Zone Number
	Independent Variable
	Dependent Variable

Notice that fieldmaps do not associate variables with the plot. In field plots, the variables are assigned to the axes, and the axes assignments are used for all fieldmaps in the current frame. The set of zones assigned to a fieldmap are dictated by the dataset. In general, for transient data the set of zones representing a particular region in space for all time is associated with a given fieldmap. For static data, each static zone is associated with a unique fieldmap.



Example 15: Setting Style for Multiple Linemaps

Given a dataset with the following properties:

- 10 Zones, (all I-ordered, all with an arbitrary number of data points, static data - i.e. independent of time)
- 4 Variables (named "Time", "Pressure", "Temperature", "Density")

Create a line plot containing three lines as follows:

- Line 1: Use zone 3, plot "Time" vs. "Pressure"
- Line 2: Use zone 5, plot "Time" vs. "Temperature"
- Line 3: Use zone 7, plot "Time" vs. "Density"

Create three linemaps and assign the zone, independent variable, and dependent variable according to the table above. Each linemap can then be assigned any style you want. The resulting plot is shown below:

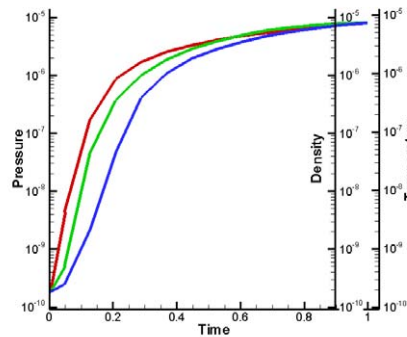


Figure 13-2. An example of three

```
StyleValue sv1( SV_LINEMAP, SV_ASSIGN, SV_YAXIS);
objectSet = 2;
sv1.set( 2, objectSet);
objectSet = 3;
sv1.set( 3, objectSet);

StyleValue sv2( SV_XYLINEAXIS, SV_YDETAIL, SV_COORDSCALE );
sv2.set( CoordScale_Log, 1);
sv2.set( CoordScale_Log, 2);
sv2.set( CoordScale_Log, 3);

StyleValue sv3( SV_XYLINEAXIS, SV_YDETAIL, SV_RANGEMAX );
sv3.set( 0.000012, 1);
sv3.set( 0.000012, 2);
sv3.set( 0.000012, 3);

StyleValue sv4( SV_XYLINEAXIS, SV_YDETAIL );
sv4.set( -10, 3, SV_AXISLINE, SV_OFFSET);
sv4.set( -3, 3, SV_TITLE, SV_OFFSET);

objectSet.assign("[1-3]");
StyleValue sv5(SV_LINEMAP, SV_LINES, SV_LINETHICKNESS);
sv5.set( 0.8, objectSet);
```



Example 16: Setting Style for Multiple Fieldmaps

Given a dataset with the following properties:

- 10 Zones, (all IJ-ordered, all with an arbitrary number of data points and static zones)
- 4 Variables (named "X", "Y", "Temperature", "Density")

Suppose we want to create a field plot showing zones 4, 5 and 9 where:

- For zone 4, show mesh
- For zone 5, show contours with continuous contour flooding
- For zone 9, show vectors and streamtraces
- For all other zones, show two surface streamtraces

Once the data is loaded and a field plot is made active, the fieldmaps associated with the zones will automatically be created and cannot be added to or deleted independently of the associated zone(s). When the above dataset is loaded we will have 10 fieldmaps at our disposal, one per static zone. The resulting plot is shown below:

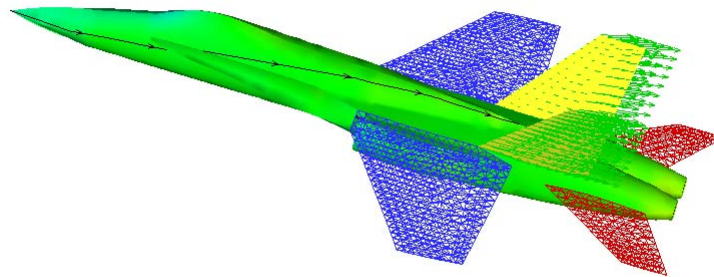


Figure 13-3. A field plot composed of several fieldmaps, each with its own style settings.

```
// Data is loaded and the plottype is set to 3D.
StyleValue sv1;
sv1.set( 4, 1, SV_GLOBALCONTOUR, SV_VAR);

StyleValue sv2(SV_FIELDLAYERS );
sv2.set( TRUE , SV_SHOWCONTOUR );
sv2.set( FALSE, SV_SHOWEDGE );
sv2.set( TRUE , SV_SHOWSHADE );

StyleValue sv3(SV_GLOBALTHREEDVECTOR);
sv3.set( 1, SV_UVAR );
sv3.set( 2, SV_VVAR );
sv3.set( 3, SV_WVAR );
```

```

sv1.set( TRUE, SV_FIELDLAYERS, SV_SHOWVECTOR );

TecUtilResetVectorLength();
sv1.set( TRUE, SV_STREAMTRACELAYERS, SV_SHOW);

StyleValue sv4(SV_FIELDMAP);
Set objectSet("[2,4-6,8,10-12]");
sv4.set( FALSE, objectSet, SV_MESH, SV_SHOW);

objectSet.assign("[1-3,7-10]");
sv4.set( FALSE, objectSet, SV_CONTOUR, SV_SHOW);

objectSet.assign("[1,3-7,9-12]");
sv4.set( FALSE, objectSet, SV_VECTOR, SV_SHOW);

objectSet.assign("[1,3-7,9-12]");
sv4.set( FALSE, objectSet, SV_SHADE, SV_SHOW);

objectSet.assign("[2,8]");
sv4.set( Yellow_C, objectSet, SV_SHADE, SV_COLOR);

TecUtilStreamtraceAdd(1, Streamtrace_SurfaceLine, StreamDir_Both,
                      8.6337699353, -0.64474074805, 1.2887317501,
                      0.0000000000, 0.00000000000, 0.0000000000);

TecUtilStreamtraceAdd(1, Streamtrace_SurfaceLine, StreamDir_Both,
                      1.6039546373, -0.46701803889, 0.20915714589,
                      0.0000000000, 0.00000000000, 0.0000000000);

sv1.set( ColorMapDistribution_Continuous, 1, SV_GLOBALCONTOUR,
        SV_COLORMAPFILTER, SV_COLORMAPDISTRIBUTION);

```

13 - 1.1 Linemap and Fieldmap Styles

Both linemaps and fieldmaps have “power switches” that turn on/off a particular style for the entire frame. They also have settings to turn on/off a style for an individual linemap or fieldmap. For example, if you want to show contours in a field plot, the contour layer must first be activated. Then, the contour show setting must be activated for the fieldmaps that are to display contour style.

Linemap and fieldmap styles are driven by the **Mapping Style** and **Zone Style** dialogs. When these dialogs are used in conjunction with the [Code Generator](#), you will be able to determine the appropriate calls to achieve the desired styles. The remainder of this chapter provides detailed information regarding the available style settings.

13 - 2 Line Plots



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

A line plot is the simplest type of graph produced by the Tecplot Engine. Each line on the line plot represents one series of data points, where each data point is defined by its independent and dependent variable values. A series of data points is referred to as a mapping (or map, for short).

Line plots are usually created from one-dimensional, I-ordered data¹. The data used for line plots must have at least two variables defined at each data point. The same number of variables must be defined at each data point.

There are two supported types of line plots: XY plots and Polar Line plots. An example of XY and Polar Line plots is shown below:

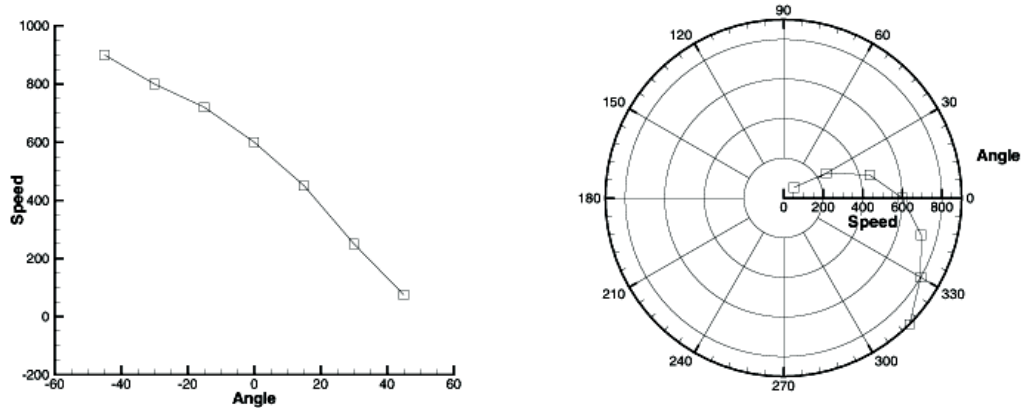


Figure 13-4. A plot of speed versus angle in XY Line (left) and Polar Line (right) plot

- **XY Plots** - XY plots are plotted on Cartesian coordinates using X and Y as the independent and dependent variables. XY plots can include line, symbol, bar and/or error bar mapping layers.
- **Polar Plots** - Polar plots are plotted on polar coordinates using Theta and R values. Polar plots can include line and/or symbol layers. Polar Line plots have the following three drawing options:
 - **Curved Lines (Theta-R Interpolation)** - The connection between two points is a curve. This may slow plotting speed for large datasets.
 - **Straight Lines (X-Y Interpolation)** - The connection between two points is a straight line.

The difference between the two Polar Line Drawing Modes is shown below:

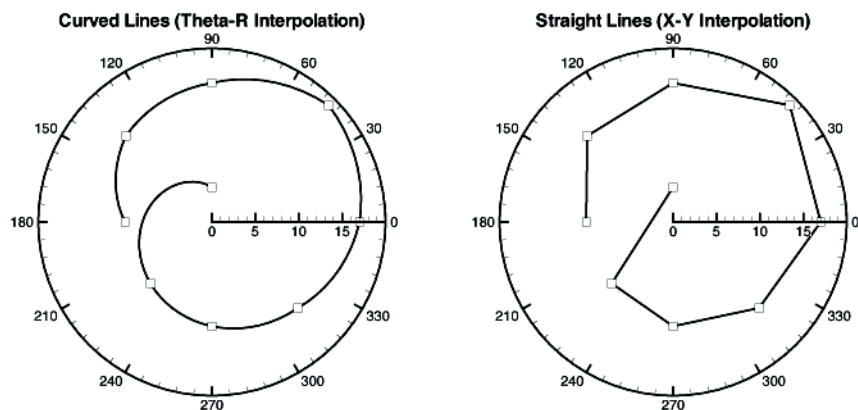


Figure 13-5. The Polar Drawing Modes: Curved lines are shown on the left and straight lines are shown on the right.

1. Refer to [Chapter 10: "Data Structure"](#) for information on Data Structure, including I,J, and K-ordering of datasets.

For information on limiting the number of points plotted for a given linemap, refer to [Section 13 - 4 “Plotting Subsets of Data”](#).

13 - 2.1 Mapping Layers

Line plots are composed of the graphs of one or more pairs of variables (XY pairs in XY Line plots or Theta-R pairs in Polar Line plots). These pairs and their dependency relations are referred to as mappings. Mappings are defined for each frame; the same dataset can have a different set of mappings in each frame it is attached to.

To work with a given mapping layer, you will need to activate the mapping layer. You can opt either to include or exclude individual linemaps from a given layer as well.

Line plots can include any combination of the following mapping layers:

- **Lines** - Lines can be drawn as linear segments or curves that fit the data points. Refer to [Section 13 - 2.3 “Curve Types”](#) for details regarding the various curve types. For each linemap, you can customize: the line color, line pattern, pattern length, and line thickness for the line layer.
- **Symbols** - Each data point is represented by a symbol. For each linemap, you can customize: the symbol shape, outline color, fill mode, fill color, symbol size, line thickness, and symbol spacing.
- **Bars** (XY only) - Each data point is represented by a vertical or horizontal bar. For each linemap, you can customize: the direction of the bars, the outline color, fill mode, fill color, bar size, and line thickness.
- **Error Bars** (XY only) - Error bars are drawn for each data point. The error bar value is determined by a third variable which must be assigned prior to activating the layer. For each linemap, you can customize: the error bar variable, the error bar type, color, size, line thickness, and spacing. An example plot with error bars is shown below.

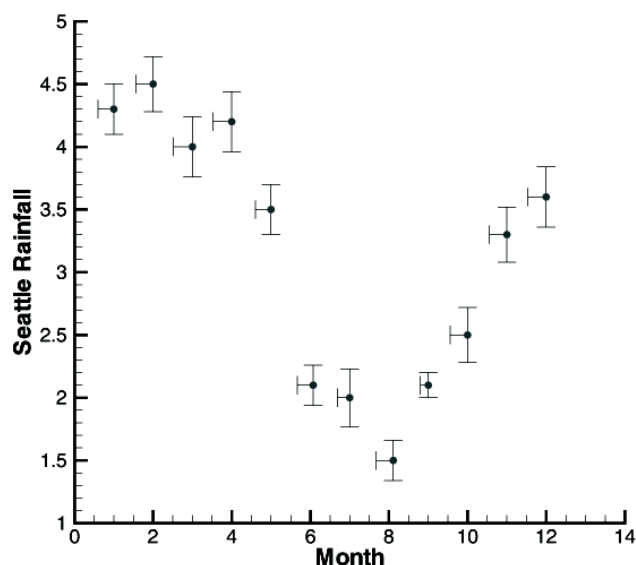


Figure 13-6. An XY Line plot with symbols and error bars.



Example 17: Changing the Line Thickness for a Set of Linemaps

To set the line thickness for linemaps 2, 3 and 5, use the following code

```
StyleValue sv(SV_LINEMAP, SV_LINES, SV_LINETHICKNESS);  
sv.set(2.5, Set("[2,3,5]"));
```

Note that this example illustrates a way to use the Set and StyleValue classes which is different from the code created by the [Code Generator](#).



If you want to change the style for all linemaps, do not supply a Set in the StyleValue::set method call.



Example 18: Setting the Line Color for all Linemaps

This sample code sets the line color for all linemaps because no linemap set is specified. If a linemap set were specified, the line color would be set only for that linemap set.

```
StyleValue sv( SV_LINEMAP, SV_LINES, SV_COLOR);  
sv.set(Green_C);
```



Example 19: Creating an XY Line Plot with Lines and Symbols

The following sample code creates an XY Line Plot with a blue line and red circles for a single mapping:

```
StyleValue sv1;  
sv1.set( TRUE, SV_STREAMTRACELAYERS, SV_SHOW);  
  
TecUtilFieldLayerSetIsActive( SV_SHOWMESH, FALSE);  
  
double xrawdata[] = { -.3203, -.0891 }; // x data for term line
```

```
double yrawdata[] = { .3829, -.4663 }; // y data for term line
TecUtilStreamtraceSetTermLine( 2, &(xrawdata[0]), &(yrawdata[0]));

TecUtilStreamtraceAdd( 1, Streamtrace_VolumeRod, StreamDir_Both,
                      0.5, 0.3311, 0.4310,
                      0.5, 0.3311, 0.4312);

StyleValue sv2(SV_STREAMATTRIBUTES);
sv2.set( TRUE, SV_TERMLINE, SV_ISACTIVE);
```



Example 20: Creating an XY Line Plot with Bars

The following sample code creates an XY Line Plot with green bars that have a blue outline:

```
TecUtilLinePlotLayerSetActive( SV_SHOWLINES , FALSE);
TecUtilLinePlotLayerSetActive( SV_SHOWBARCHARTS, TRUE);
```

13 - 2.2 Line Plot Axes

XY Line plots can have up to five X-axes and five Y-axes simultaneously. Polar Line plots can have only one Theta-axis and only one R-axis. When you first load an ordered dataset, some mappings are automatically created for you. If your dataset has more than two variables, mappings are created that associate the first variable with each of the other variables for the first zone only.

You can alter the axis assignments using the StyleValue class. Refer to [Section 13 - 6 "Axes"](#) for additional details on the Axes style settings available.



Example 21: Creating an XY Line Plot with Multiple Axes

The following sample code creates an XY Line Plot with 3 mappings and 2 y-axes:

```
TecUtilLineMapCreate();
Set objectSet(2);
StyleValue sv(SV_LINEMAP);
sv.set( "Map 2", objectSet, SV_NAME);
sv.set( 1 , objectSet, SV_ASSIGN , SV_XAXISVAR);
sv.set( 3 , objectSet, SV_ASSIGN , SV_YAXISVAR);
sv.set( 1 , objectSet, SV_ASSIGN , SV_ZONE);
sv.set( Lines_I, objectSet, SV_INDICES, SV_IJKLINES);
sv.set( 2 , objectSet, SV_ASSIGN , SV_YAXIS);
TecUtilLineMapSetActive(objectSet.getRef(), AssignOp_PlusEquals);

TecUtilLineMapCreate();
```

```

objectSet = 3;
sv.set( "Map 3", objectSet, SV_NAME);
sv.set( 1      , objectSet, SV_ASSIGN, SV_XAXISVAR);
sv.set( 2      , objectSet, SV_ASSIGN, SV_YAXISVAR);
sv.set( 1      , objectSet, SV_ASSIGN, SV_ZONE);
sv.set( Lines_I, objectSet, SV_INDICES, SV_IJKLINES);
TecUtlLineMapSetActive( objectSet.getRef(), AssignOp_PlusEquals);

```

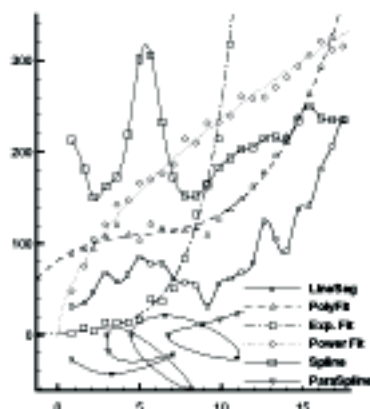
13 - 2.3 Curve Types

A variety of curve fits and spline fits are also offered. By specifying the curve type, you control how the data points are connected.

The following curve types are available:

- **Line Segments** A series of linear segments connect adjacent data points.
- **Linear Fits** A linear function is fit to the data points. In XY Line plots, this will be a straight line.
- **Polynomial Curve Fits** A polynomial of order N is fit to the data points (where $1 \leq N \leq 10$, for $N=1$ a Linear Fit is done).
- **Exponential Curve Fits** An exponential curve fit finds the best curve of the form $Y=e^{b \cdot X} + c$ (equivalent to $Y=a \cdot e^{b \cdot X}$, where $a = e^c$). To use this curve type, Y-values for this variable must be all positive or all negative. If the function dependency is set to $X=f(Y)$ all X-values must be all positive or all negative.
- **Power Curve Fits** A power curve fit finds the best curve of the form $Y=e^{b \cdot \ln X} + c$ (equivalent to $Y=a \cdot X^b$, where $a = e^c$). To use this curve type, Y-values for this variable must be all positive or all negative; X-values must be all positive. If the function dependency is set to $X=f(Y)$, X-values must be all positive or all negative, and the Y-values must all be positive.
- **Splines** A smooth curve is generated that goes through every point. The spline is drawn through the data points after sorting the points into increasing values of the independent variable, resulting in a single-valued function of the independent variable. The spline may be clamped or free. With a clamped spline, you supply the derivative of the function at each end point; with a non-clamped (natural or free) spline, these derivatives are determined for you. In XY Line plots, specifying the derivative gives you control over the initial and final slopes of the curve.
- **Parametric Splines** Creates a smooth curve as with a spline, except the assumption is that both variables are functions of the index of the data points. (For example, in XY Line plots, ParaSpline fits $x=f(i)$ and $y=g(i)$ where $f(i)$ and $g(i)$ are both smooth.) This spline may result in a multi-valued function (of either or both axis variables).
- **Extended Curve Fit** Uses a curve fit.

Linear Fit, Polynomial Fit, Exponential Fit, and Power Fit are all determined by using a least squares algorithm. Examples of each curve fit type are shown below:



Example 22: Adding a Polynomial Curve Fit

The following sample code modified linemap 3 in an existing XY Line plot to use a polynomial curve fit (order=5):

```
Set objectSet(1);
StyleValue LineMapStyle(SV_LINEMAP, SV_CURVES);
LineMapStyle.set(CurveType_PolynomialFit, objectSet,
SV_CURVETYPE);
LineMapStyle.set(5, objectSet, SV_POLYORDER);
```

13 - 2.4 Line Legends

You can generate a legend that shows the line and symbol attributes of the mappings. In XY Line plots, this legend includes the bar chart information. The legend can be positioned anywhere within the line plot frame. By default, all mappings are shown. However, redundant entries are removed from the display.

Line legends have the following options:

- **Show Mapping Names** Opt whether to include mapping names in the legend.
- **Text** Format the text for the legend by choosing a color, font, text height, and line spacing between entries in the dialog.
- **Position** The legend is automatically placed for you. You may specify the position of the legend using X as a percentage of the frame width and Y as a percentage of the frame height.
- **Anchor** By default, the legend is anchored in the top right. Use the anchor parameter to adjust this setting.
- **Legend Box** You can opt to display a box around the line legend. You can customize the line legend box with the following settings:

- **Box Style** Specify whether to have no box, a filled box, or an outline.
- **Box Line Style** If you set the box style to filled or outline, you can customize the box thickness and line color. Any of the basic color values are available. Refer to [Section 13 - 5.1 “Basic Colors”](#) for details.
- **Fill Color** If you set the box style to filled, specify the fill color using any of the basic color values. Refer to [Section 13 - 5.1 “Basic Colors”](#) for details.
- **Margin** Use the margin to specify the space between the legend and the legend box. The margin is available for all box types.



Example 23: Activating the Line Legend

The following sample code illustrates turning on the line legend for an existing line plot:

```
StyleValue sv(SV_GLOBALLINEPLOT, SV_LEGEND, SV_SHOW);
sv.set(TRUE);
```

13 - 3 Field Plots



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

There are two types of field plots available: 2D and 3D Cartesian. The axes in field plots are all independent variables.

You can create a 2D or 3D field plot by using any combination of the following plot properties:

- [Fieldmap Layers](#) - By default, 2D and 3D field plots are initially displayed with Mesh and Edge fieldmap layers.
- [Plotting Subsets of Data](#) - Points or surfaces may be selected as the source for your data points.
- [Plot Effects](#) - 3D field plots may be enhanced with Lighting effects and Translucency.
- [Derived Objects](#) - Field plots may also contain any combination of iso-surfaces, slices, and streamtraces (which are derived from the values in the dataset).

13 - 3.1 Fieldmap Layers

A layer is a way of representing a frame's dataset. The complete plot is the sum of all the active layers, axes, text, geometries, and other elements added to the data plotted in the layers. This section describes the six fieldmap layers for 2D and 3D Cartesian plot types.

To work with a given fieldmap layer, you will need to activate the fieldmap layer. You can opt either to include or exclude individual fieldmaps from a given layer as well.

The following fieldmap layers are available:

- [Mesh](#)

- [Contour](#)
- [Vector](#)
- [Scatter](#)
- [Shade](#)
- [Edge](#)

Mesh

The Mesh plot layer displays the lines connecting neighboring data points within a fieldmap. For each fieldmap, you can customize the following mesh settings:

- **Show** Opt whether to display the mesh for the given set of fieldmaps.
- **Mesh Type** There are three mesh types available:
 - **Wire Frame** Wire frame meshes are drawn below any other zone layers on the same zone. In 3D Cartesian plots, no hidden lines are removed. For 3D volume zones (finite-element volume or IJK-ordered), the full 3D mesh (consisting of all the connecting lines between data points) is not generally drawn because the sheer number of lines would make it confusing. The mesh drawn will depend upon your choice of “Surfaces to Plot”. See [Section 13 - 4.2 “Surfaces”](#) for further details. By default, only the mesh on exposed cell faces is shown.
 - **Overlay** Similar to Wire Frame, mesh lines are drawn over all other zone layers except for vectors and scatter symbols. In 3D Cartesian plots, the area behind the cells of the plot is still visible (unless another plot type such as contour flooding prevents this). As with Wire Frame, the visibility of the mesh is dependent upon your choice of “Surfaces to Plot”. See [Section 13 - 4.2 “Surfaces”](#) for further details.
 - **Hidden Line** Similar to Overlay, except hidden lines are removed from behind the mesh. In effect, the cells (elements) of the mesh are opaque. Surfaces and lines that are hidden behind another surface are removed from the plot. For 3D volume zones, using this plot type obscures everything inside the zone. If you choose this option for 3D volume zones, then choosing to plot every surface has the same effect as plotting only exposed cell faces, but is much slower.



The opaque surfaces created by Hidden Line are not affected by the Lighting effect (there is no light source shading). However, it is affected by translucency.

- **Mesh Color** Specify the mesh color using any of the coloring options described in [Section 13 - 5 “Coloring”](#).
- **Mesh Line** You can customize the mesh line pattern and thickness.



Example 24: Creating a Custom Mesh Plot

The following sample code creates a Mesh Plot with dashed blue lines:

```
Set objectSet(1);
StyleValue FieldMapStyle(SV_FIELDMAP, SV_MESH);
FieldMapStyle.set( LinePattern_Dashed, objectSet, SV_LINEPATTERN);
FieldMapStyle.set( Blue_C, objectSet , SV_COLOR);
```



Example 25: Changing the Mesh Color

The following sample code illustrates how to set the mesh color for fieldmaps 1 and 3:

```
StyleValue sv( SV_FIELDMAP, SV_MESH, SV_COLOR);
sv.set( Blue_C, Set("[1,3]"));
```



If you want to change the style for all fieldmaps, do not supply a Set in the MeshColor.Set method call.

Contour

Contour plots can be used to show the variation of one variable across the data field.



Contour plots can only be plotted with organized data, such as IJ-ordered, IJK-ordered, or FE-data. Refer to [Section 3 - 6 "Working with Unorganized Datasets"](#) in the [User's Manual](#) for information on organizing your dataset.

There are two levels of control for contour plots: control by contour group number, or control for a given fieldmap. A contour group is used to link a group of settings such as contour level values and contour labels to a given contour variable. You can use up to eight different contour groups.

Before activating the contour layer, you should assign a variable to at least contour group 1. If you plan to work with multiple contour groups, assign a contour variable to those groups, as well.

Fieldmap Settings

The following contour style settings can be set for a given fieldmap or group of fieldmaps:

- **Show/Hide** You can opt whether or not to display contours for a given fieldmap.
- **Contour Type** You can create contour plots of five different types (An example of each plot type is shown below):
 - **Lines** Lines of constant value are drawn for the specified contour variable.

- **Flood** Floods regions between contour lines with colors from the global color map. The distribution of colors used for contour flooding may be banded or continuous. When banded distribution is used for flooding, a solid color is used between contour levels. If continuous color distribution is used, the flood color will vary linearly, as defined by the colormap.
- **Both Lines and Flood** Combines the above two options.
- **Average Cell** Floods cells or finite-elements with colors from the global color map according to the average value of the contour variable over the data points bounding the cell.
- **Primary Value** Floods cells or finite-elements with colors from the global color map according to the primary value of the contour variable for each cell.

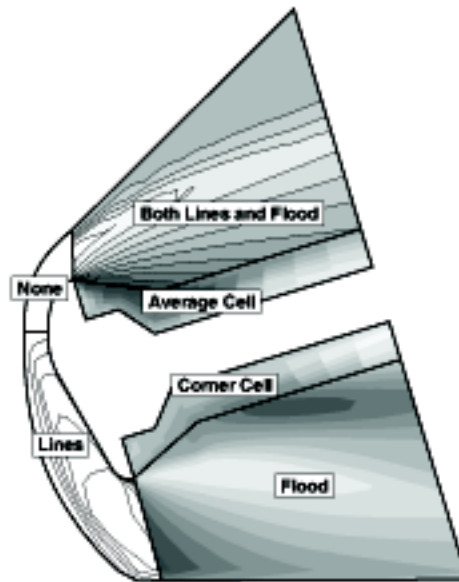


Figure 13-7. Contour types.

- **Flood By** Specify a contour group (C1, C2, C3, C4, C5, C6, C7, or C8) or assign variables to the RGB color map. NOTE: You must assign RGB variables prior to using RGB Coloring. Refer to [Section 13 - 5.3 "RGB Multi-coloring"](#) for details.
- **Lines By** Specify which contour group identifies the contour lines (applicable only when the contour type is 'Lines' or both 'Lines and Flood').
- **Line Attributes** When the contour type is set to Lines or Lines and Flood, you can customize the line color, line pattern and the pattern length.
- **Use Lighting (3D only)** Turn the lighting effects on or off. If lighting is on, the contour coloring may not match the contour legend because of shading effects.



You can also turn the contour layer on and off for a subset of fieldmaps.



Example 26: Changing the Contour Type

The following sample code illustrates setting the contour type to 'Lines and Flood' for all fieldmaps:

```
StyleValue sv( SV_FIELDMAP, SV_CONTOUR, SV_CONTOURTYPE);  
sv.set(ContourType_Overlay);
```

Contour Group Settings

The following contour style settings can be set for a given contour group:

- **Contour Levels** A contour level is a value at which contour lines are drawn, or for banded contour flooding, the border between different colors of flooding. You can add, change, or delete any of the contour levels assigned to a given group. Default values are calculated for you based on the dataset.
- **Color Map** Although the color map is global (affecting all frames), there are some adjustments you can make that apply only to a contour group in the current frame. Specify the color map group to use for contour coloring. See [Section 13 - 5 "Coloring"](#) for more information on color map groups.



Example 27: Changing the Active Colormap

The following sample code demonstrates turning off colormap linking and assigning a different colormap to contour group 3:

```
TecUtilFieldLayerSetActive( SV_SHOWCONTOUR, TRUE);  
  
StyleValue sv;  
//Assign var 1 to Contour Group 3  
sv.set(1, 3, SV_GLOBALCONTOUR, SV_VAR);  
  
// Unlink the colormaps  
sv.set(FALSE, SV_GLOBALLINKING, SV_LINKCOLORMAPS);  
  
// Assign TwoColor to ColorMap group 3.  
sv.set(ColorMap_TwoColor, 3, SV_GLOBALCOLORMAP, SV_CONTOURCOLORMAP);  
  
// Tell zone 1 to use contour group 3 for flooding.  
Set zset(1);  
sv.set(ContourColoring_Group3, zset, SV_FIELDMAP,  
       SV_CONTOUR, SV_FLOODCOLORING);
```

• Color Distribution Methods

- **Banded** - A solid color is assigned for all values within the band between two levels.
- **Continuous** - The color distribution assigns linearly varying colors to all multi-colored objects or contour flooded regions. You can vary the default assignment of colors by entering a “Min” or “Max” value for Color Map Endpoints.



Limitation for Continuous Coloring

Vector-based export files such as WMF cannot show continuous flooding. Objects that use continuous flooding are reduced to contain average cell flooding where each cell is flooded a solid color based on the averages of the values at each vertex.

- **Use Approximate Continuous Flooding** - Causes each cell to be flooded using interpolation between the RGB values at each node. When the transition from a color at one node to another node crosses over the boundary between control points in the color spectrum, approximate flooding may produce colors not in the spectrum. Choosing not to use this option is slower, but more accurate.
 - **Color Cutoff** - Specify a range within which contour flooding and multi-colored objects (such as scatter symbols) are displayed.
 - **Contour Labels** - You can also add contour labels to your plot, where labels are placed either by contour number or value.
-



Example 28: Adding Contour Labels

The following sample code adds contour labels to the plot. NOTE: the labels will be automatically positioned on the plot.

```
StyleValue sv;
sv.set(TRUE, 1, SV_GLOBALCONTOUR, SV_LABELS, SV_SHOW);
TecUtilFieldLayerSetIsActive( SV_SHOWCONTOUR, TRUE);
```

Contour Legends

You have the option to add a contour legend to your plot. Contour Legends can be added on a group by group basis. You can customize the following attributes of your contour legend:

- **Show/Hide** - You can opt whether or not to display a contour legend for each contour group.
- **Labels** - You can customize both the header label and number labels in your contour legend. The following settings are available:
 - **Header Label** - The header label is used to display the contour variable name in the legend. You can opt whether to display a header label, and also specify its font type and height.

- **Label Color** - You can set the color for both the header and number labels to any of the basic color provided by the Tecplot Engine. Refer to [Section 13 - 5.1 "Basic Colors"](#) for a list of available options. NOTE: the same color will be used for both types of labels.
- **Level Labels** - The Level Labels are used to identify the numerical value of each contour level. You can customize the font and number format used for the level labels.
- **Color Bands** - You can opt whether or not to display separators at each contour level. This setting is available for both continuous and banded coloring.
- **Size and Position** - You can customize the overall size and position of the contour legend with the following options:
 - **Resize Automatically** - Set Resize Automatically to TRUE to have the Tecplot Engine manage the size of your legend based on the number of levels to display.
 - **Line Spacing** - Set the line spacing to specify the distance between level labels. This does not change the number of entries in the legend, so a large value here creates a large legend. Use Level Skip to reduce the number of entries in the legend.
 - **Position (X%/Y%)** - Specify the placement of the anchor position as a percentage of frame height. The location of the anchor or legend origin can be set to top-middle, bottom-left, middle-right, etc.
 - **Alignment** - You can opt to display either a vertical or horizontal legend.
- **Levels to Display** - You can customize the contour levels displayed in the legend with the following settings:
 - **Level Skip** - Specify the number of levels between numbers on the legend. This also adjusts the number of levels between any contour labels on the plot. Skipping levels on the contour legend compresses the color bar (if one appears); it does not change the spacing between text entries on the legend.
 - **Cutoff Levels** - Specify whether or not to display the cutoff level settings in the legend.
 - **Contour Label Location** - Use the Contour Label Location to specify whether to label the contour at: the contour levels, color map divisions, or a specified increment.
- **Legend Box** - You can opt to display a box around the contour legend. The contour legend box can be customized with the following settings:
 - **Box Style** - Specify whether to have no box, a filled box, or an outline.
 - **Box Line Style** - If you set the box style to filled or outline, you can customize the outline thickness and color. Any of the basic color values are available. Refer to [Section 13 - 5.1 "Basic Colors"](#) for details.
 - **Fill Color** - If you set the box style to filled, specify the fill color using any of the basic color values. Refer to [Section 13 - 5.1 "Basic Colors"](#) for details.
 - **Margin** - Use the margin to specify the space between the legend and the legend box. The margin is available for all box types.



Example 29: Adding a Contour Legend

The following sample code adds a contour legend to an existing field plot:

```
TecUtilFieldLayerSetActive( SV_SHOWCONTOUR, TRUE);

StyleValue sv1(SV_GLOBALCONTOUR);
sv1.set( 3, 1, SV_VAR);
sv1.set( TRUE, 1, SV_LEGEND, SV_SHOW);
sv1.set( FALSE, 1, SV_LEGEND, SV_ISVERTICAL);
sv1.set( FALSE, 1, SV_LEGEND, SV_OVERLAYBARGRID);
```

Vector

You can create vector plots by activating the Vector layer, and specifying the vector component variables.



You must set the vector variables prior to activating the Vector layer. 2D plots require U and V variables, while 3D plots require U, V, and W variables.

Some vector settings can be customized on a fieldmap by fieldmap level, while other settings are global to all vectors in the frame.

Fieldmap Settings

You can control any of the following vector plot attributes on a fieldmap by fieldmap level:

- **Show/Hide** - Opt whether to display the vector layer for a fieldmap or set of fieldmaps.
- **Vector Type** - Select from the following arrowhead locations to specify the vector type: tail at point, head at point, anchor at midpoint, or head only. [Figure 13-8](#) shows examples of each of the vector plot types.

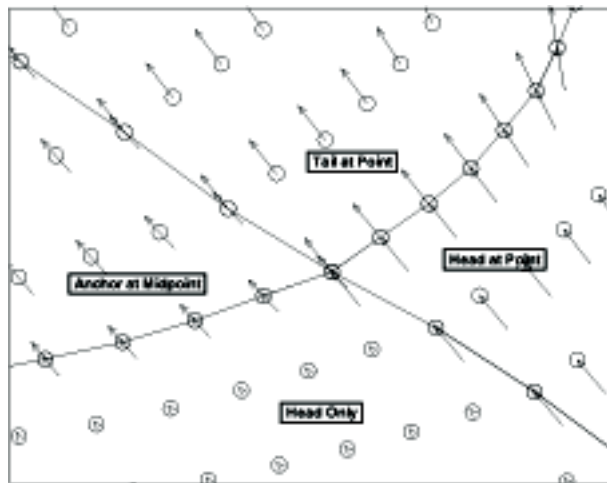


Figure 13-8. The Vector plot types: tail at point, head at point, anchor at midpoint, and head only.

- **Arrowhead Style** - Specify the style of arrowheads to use. [Figure 13-9](#) displays the available arrowhead styles.
 - **Plain (default)** - Line segments drawn from the head of the vector.
 - **Filled** - Filled triangles with apex at the head of the vector.

- **Hollow** - Hollow triangles with apex at the head of the vector.

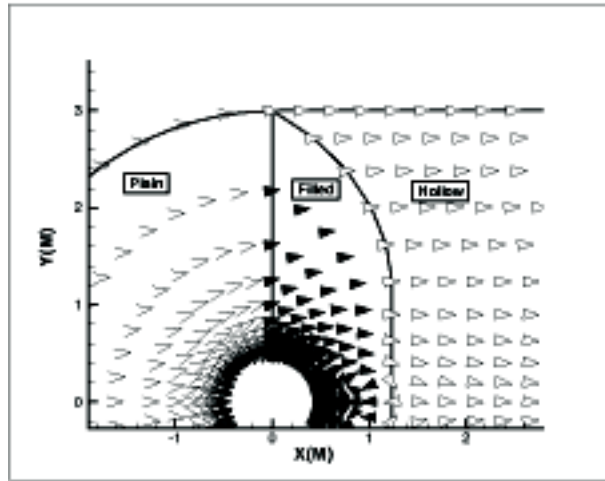


Figure 13-9. Arrowhead types for vector plots (plain, filled and hollow).

- **Vector Tangent** - Specify whether to display the vectors as 3D vectors with both the normal and tangent components or just the tangents components. Tangent vectors are drawn on 3D surfaces only where it is possible to determine a vector normal to the surface. Tangent vectors are not drawn on a plot where multiple surfaces intersect each other using common nodes because there is more than one normal to choose from (e.g. a volume IJK-ordered zone where both the I and J-planes are plotted). If tangent vectors cannot be drawn, standard vectors are plotted instead.

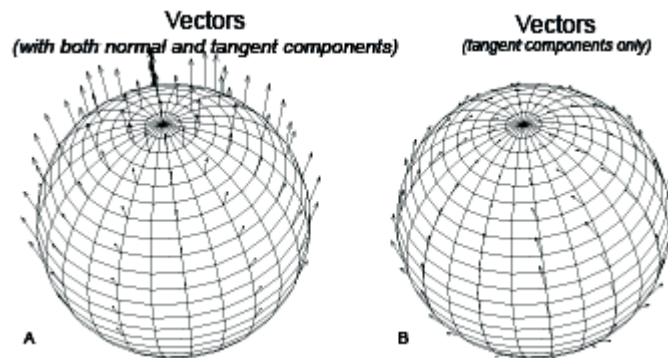


Figure 13-10. Comparison of the Vector Tangent options. (A) Vectors are drawn with both the normal and tangent components. (B) Vectors are drawn with only the tangent components.

- **Vector Line** - You can customize the vector line pattern, line pattern length, line color, and line thickness. Refer to [Section 13 - 5 "Coloring"](#) for details on the coloring options available.



Example 30: Creating a Custom Vector Plot

The following sample code illustrates how to create a vector plot with hollow arrows placed at the point, where the vector variables (U,V, and W) are variables 4, 5 and 6, respectively:

```
StyleValue sv1(SV_FIELDLAYERS);
sv1.set( TRUE, SV_SHOWVECTOR);
TecUtilResetVectorLength();

Set objectSet(1);
StyleValue sv2( SV_FIELDMAP, SV_VECTOR);
sv2.set( Arrowhead_Hollow      , objectSet, SV_ARROWHEADSTYLE);
sv2.set( VectorType_HeadAtPoint, objectSet, SV_VECTORTYPE);
```

Global Vector Settings

The following attributes are global to all vectors:

- Arrowhead angle and size
- Vector length
- **Reference Vector** - A reference vector is a vector of specified magnitude placed on the plot as a measure against all other vectors. [Figure 13-11](#) shows a plot with a reference vector.

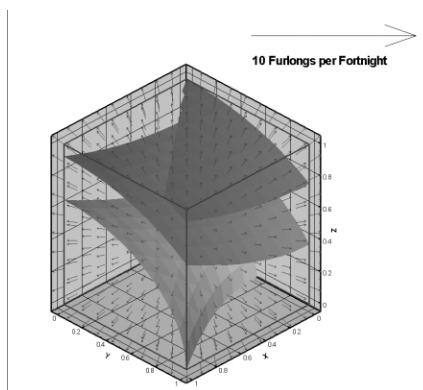


Figure 13-11. An example of a vector plot with a reference vector included

Scatter

Scatter plots are plots of symbols at the data points in a field. The symbols may be sized according to the values of a specified variable, colored by the values of the contour variable, or uniformly sized or colored. Unlike contour plots, scatter plots do not require any mesh structure connecting the points. This allows you to make scatter plots of unorganized data.



The Scatter variable must be set prior to activating the Scatter layer.

You can control some scatter settings on a fieldmap by fieldmap basis, while other settings are global to the current frame. Refer to the remainder of this section for details.

Fieldmap Settings

You can control any of the following scatter plot attributes on a fieldmap by fieldmap level:

- Show or hide the scatter layer for a given set of fieldmaps.

- Symbol shape
- Outline color
- Fill mode and color
- Scatter size
- Line thickness



Example 31: Creating a Custom Scatter Plot

The following sample code generates a Scatter Plot with purple circles, line thickness of 0.2:

```
TecUtilFieldLayerSetIsActive(SV_SHOWSCATTER, TRUE);

Set objectSet(1);
TecUtilZoneSetScatterSymbolShape( SV_GEOMSHAPE,
                                   objectSet.getRef(),
                                   GeomShape_Circle);

StyleValue sv2( SV_FIELDMAP, SV_SCATTER);
sv2.set( Purple_C, objectSet, SV_COLOR);
sv2.set( 0.2,      objectSet, SV_LINETHICKNESS);
sv2.set( 1.0,      objectSet, SV_FRAME_SIZE);
```

Global Settings

The following attributes are global to all fieldmaps displaying the scatter layer:

- **Scatter Size Variable** - If at least one fieldmap is set to “size the scatter symbol according to a variable”, you can customize and set a scatter variable.
- **Reference Scatter Symbol** - If you are using a scatter-size variable, it is sometimes useful to create a reference scatter symbol that shows the size at which a data point of a given

magnitude will be represented. [Figure 13-12](#) shows a scatter plot with a reference scatter symbol.

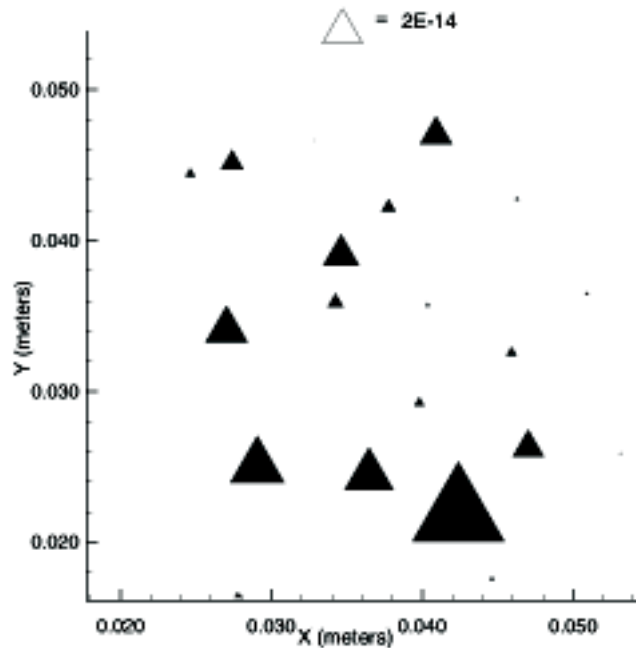


Figure 13-12. Scatter plot with reference scatter symbol.

- **Scatter legend** - You may opt to turn the scatter legend on or off, and customize its settings and position.

Shade

Although most commonly used with 3D surfaces, shade plots can also be used to flood 2D plots with solid colors, or light source shade the exterior of 3D volume plots. In 3D plots, fieldmap effects (translucency and lighting) cause color variation (shading) throughout the fieldmap(s). Shading can also help you discern the shape of the plot. See also [Section 13 - 3.2 "Plot Effects"](#).

You can control any of the following shade plot attributes on a fieldmap by fieldmap level:

- Show or hide the shade layer for a given group of fieldmaps.
- Turn lighting zone effect on or off.
- Set Shade color

Edge

An edge plot layer displays the connections of the outer lines (IJ-ordered zones), finite-element surface zones, or planes (IJK-ordered zones). The Edge layer allows you to display the edges (creases and borders) of your data. Zone borders exist only for ordered zones, or 2D finite-element zones.

There are two types of edges: creases and borders. An edge border is the boundary of a zone. An edge-crease appears when the inside angle between two cells is less than a user-defined limit. The inside angle can range from 0-180 degrees (where 180 degrees indicates coplanar surfaces). You can control the inside

angle on a global level. The default inside angle for determining an edge-crease is 135 degrees. Only edge-borders are displayed, by default.



For 2D plots, only edge-borders are available, and for FE-volume zones, only edge-creases are available.

You can control any of the following edge layer attributes:

- Show or hide the edge layer for a given group of fieldmaps.
- Edge type (borders, creases or both)
- Show or hide the corresponding index border: None, Min, Max, or Both (Min and Max).
- Edge color
- Edge line thickness



Example 32: Creating a Custom Edge Plot

The following sample code creates an Edge Plot with an inside angle of 110 degrees, edge color of green:

```
Set objectSet(1);
StyleValue FieldMapEdgeLayerStyle( SV_FIELDMAP, SV_EDGELAYER);
FieldMapEdgeLayerStyle.set(1, objectSet, SV_EDGETYPE);
FieldMapEdgeLayerStyle.set(Green_C, objectSet, SV_COLOR);
FieldMapEdgeLayerStyle.set(0, objectSet, SV_JBORDER);

StyleValue sv;
sv.set( 110.0, SV_GLOBALEDGE, SV_MINCREASEANGLE);
```

13 - 3.2 Plot Effects

For 3D plots, shade and contour fieldmap layers can be enhanced using Lighting and Translucency effects (referred to collectively as the “3D fieldmap effects”).

Lighting

There are two types of lighting effects: Paneled and Gouraud.

- **Paneled** - Within each cell the color assigned to each area by shading or contour flooding is tinted by a shade constant across the cell. This shade is based on the orientation of the cell relative to your 3D light source.
- **Gouraud** - This plot type offers a more continuous and much smoother shading than Paneled shading, but it also results in slower plotting and larger print files. Gouraud shading is not continuous across fieldmap boundaries, unless face neighbors are specified. Gouraud shading is not available for finite-element volume fieldmaps when blanking is included. A finite-

element volume fieldmap set to use Gouraud shading will revert to Paneled shading when blanking is included.



IJK-ordered data with Surfaces to Plot set to Exposed Cell Faces, faces exposed by blanking will revert to Paneled shading.

Translucency

When a fieldmap is translucent, you may view objects inside or beyond the fieldmap. All surfaces in 3D Cartesian plots may be made translucent. A different translucency may be assigned to individual fieldmaps, and may also be assigned to derived objects such as slices, streamtrace ribbons or rods, and iso-surfaces.



Example 33: Working with Translucency

The following sample code modifies fieldmap 1 to use Gouraud shading and changes the light source to have 50% intensity:

```
TecUtilContourSetVariable(4);

Set objectSet(1);
TecUtilFieldLayerSetActive( SV_SHOWCONTOUR, TRUE );
StyleValue sv1(SV_FIELDMAP, SV_EFFECTS, SV_LIGHTINGEFFECT);
sv1.set( LightingEffect_Gouraud, objectSet );

StyleValue sv2(SV_GLOBALTHREED, SV_LIGHTSOURCE);
sv2.set( 50.0, SV_INTENSITY);
```

13 - 3.3 Derived Objects

You may derive objects from your plot to view additional information about your data. You can derive streamtraces, iso-surfaces, and slices from your plot. NOTE: When you add derived objects to your plot, they are calculated based on the active fieldmaps and/or linemaps only. Refer to [Section 13 - 11 "Working with Transient Data"](#) for details. The following types of derived objects are available:

- [Streamtraces](#)
- [Iso-surfaces](#)
- [Slices](#)

Streamtraces

A streamtrace is the path traced by a massless particle placed at an arbitrary location in a steady-state vector field. Streamtraces may be used to illustrate the nature of the vector field flow in a particular region

of the plot. Because streamtraces are dependent upon a vector field, you must define vector components before creating streamtraces.



While the vector variables must be defined prior to activating streamtraces, it is not necessary to activate the Vector fieldmap layer to use streamtraces.

You can opt to display streamtraces on a fieldmap by fieldmap basis. There are two main categories of streamtraces:

- **Surface Line Streamtraces (or streamlines)** - Surface streamtraces are confined to the surface on which they are placed. They can only be placed in fieldmaps displayed as a 2D or 3D surface. When surface streamtraces are placed on a no-slip boundary surface¹, they will propagate according to the flow field very near the surface.
- **Volume Streamtraces** - Volume streamtraces can be created in 3D volume fieldmaps only (IJK-ordered or FE-volume fieldmaps). Volume streamtraces are subdivided into three categories: Volume Lines, Volume Ribbons and Volume Rods.

You can customize the following streamtrace attributes:

- **Line Styles** - Surface streamtraces or streamlines are confined to the surface on which they are placed. They can only be placed in zones displayed as a 2D or 3D surface. If you try to place streamlines in a zone displayed as a 3D volume, no streamlines are drawn.

You cannot customize streamtraces until after the streamtraces have been drawn. Once streamtraces have been drawn, you can customize the following line settings:

- **Show Path** - You can opt whether or not to display the streamtrace path or line. Set this option to FALSE, if you would like only streamtrace markers to be displayed.
- **Line Color** - You may set the color to any of the coloring options provided by the Tecplot Engine. NOTE: If you wish to use Multi-coloring and the contour variable is not currently defined, you will need to define it first. Refer to [Section 13 - 5 "Coloring"](#) for additional details.
- **Line Thickness** - Specify the thickness of the streamtrace lines.
- **Arrows** - You can opt whether or not to include arrowheads on your streamtrace lines. Arrows are not available for volume ribbons or volume rods. You can control the following attributes of the displayed arrows:
 - **Arrowhead Size** - The arrowhead size is specified as a percentage of the frame height.
 - **Arrowhead Spacing** - Specify the distance between arrowheads in terms of Y-frame units. A value of ten percent will space arrowheads approximately ten percent of the frame height apart from each other along each streamline.
- **Rod/Ribbon Styles** - If you are using volume ribbons or volume rods, you can customize the following attributes of your streamtraces:
 - **Rod/Ribbon Width** - Specify the rod or ribbon width in grid units. If you want two sets of streamtraces with different widths, you must create them individually by first creating a set of streamtraces with a specific width, and then extracting it as a zone. You can then configure a new set of streamtraces with the second width.
 - **Rod Points** - Volume rods have a polygonal cross-section. Use this parameter to specify the number of points in cross-section. (Three is an equilateral triangle, four is a square, five is a regular pentagon, and so on.) If you want two sets of volume rods with different cross-sections, you must first create one set and then extract them as zones. You can then configure a new set of streamtraces with the second cross-section.

1. Refer to the [User's Manual](#) for background information on no-slip boundary surfaces.

- **Mesh** - You can opt to display a mesh on the volume streamtraces. You specify both the line thickness for the mesh and the mesh color. Refer to [Section 13 - 5 "Coloring"](#) for additional details on the coloring options.
- **Contour Flooding** - You can opt to display contour flooding on your volume streamtrace. If you select this option, you will also need to specify which contour group to use.
- **Shade** - You can display shading on your volume streamtrace as well. You can customize the following shading effects:
 - **Shade Color** - Specify a shade color from the basic color palette. Multi-color and RGB coloring are not available (use contour flooding instead). Refer to [Section 13 - 5.1 "Basic Colors"](#) for additional details.
 - **Lighting Effect** - You can opt whether or not to include lighting on the streamtraces. If you opt to include lighting, select from "Paneled" or "Gouraud" shading.
 - **Surface Translucency** - Use the Surface Translucency setting to customize the translucency of the volume streamtraces.
- **Stream Markers** - Stream markers are symbols plotted along streamtrace paths to identify the positions of particles at certain times. [Figure 13-13](#) shows a plot with both streamtrace markers and dashes.

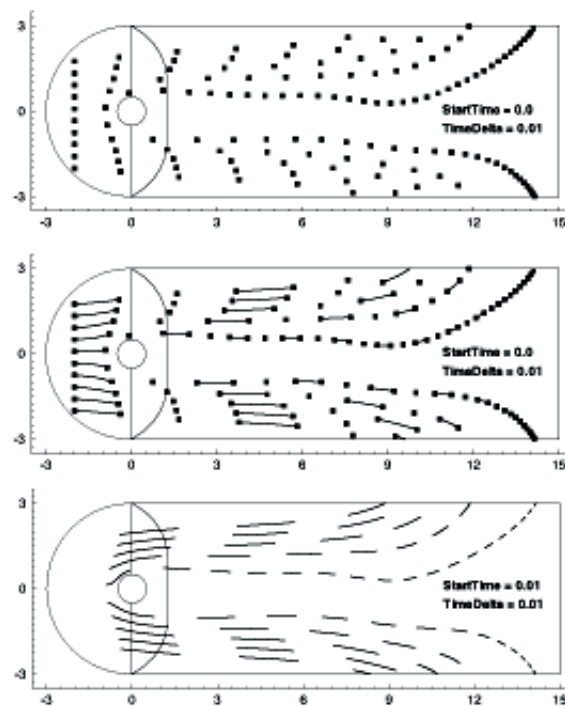


Figure 13-13. Streamtrace markers (top), dashes (bottom), and both (middle).

The spacing between stream markers is proportional to the magnitude of the local vector field. You can adjust the spacing between stream markers by specifying the time interval (or delta) between stream markers. Increasing the delta time will increase the space between stream markers and vice versa. The actual spacing is the product of the local vector magnitude and the specified delta.

You may also specify the shape, size, and color of your stream marker(s).

- **Termination Line** - A streamtrace termination line is a polyline that terminates any streamtraces that cross it. The termination line is useful for stopping streamtraces before they spiral or stall. [Figure 13-14](#) shows the cylinder data with some streamtraces terminated by a 2D streamtrace termination line.

Streamtraces are terminated whenever any of the following occur:

- The maximum number of integration steps is reached.
- Any point where the streamtrace passes outside the available data.
- The streamtrace reaches a point where the velocity magnitude is zero.
- Hits the termination line.

In 2D Cartesian plots, the termination line is drawn in the grid coordinate system and moves with the data as you zoom and translate. If you rotate a 3D dataset after drawing a streamtrace termination line, streamtraces previously terminated by the termination line may be terminated at different places, or not terminated at all if the rotated streamtrace no longer intersects the termination line.

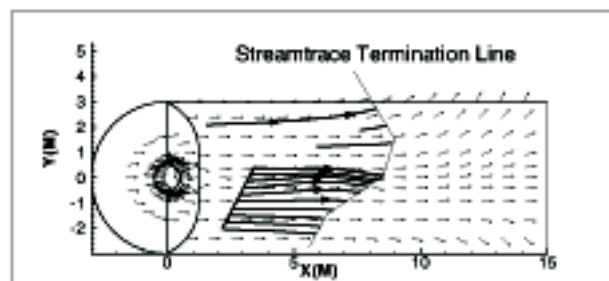


Figure 13-14. A streamtrace termination line drawn through surface streamlines (created with demo file *cylinder.plt*).

[Figure 13-14](#) shows a 3D volume plot with streamribbons and a streamtrace termination line, and how the termination points vary as the plot is rotated. Notice that the termination line itself remains in place on the screen as the plot is rotated.

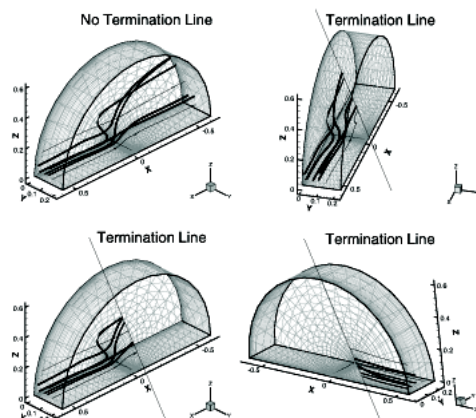


Figure 13-15. Volume streamtraces with a termination

- **Streamtrace Integration** - The Tecplot Engine uses an adaptive, step-size, trapezoidal integration algorithm to calculate streamtraces. This creates the streamtrace by moving it in a series of small steps from the starting point in the direction of (or in opposition to) the local vector field. Each step is only a fraction of a cell or element. The Tecplot Engine automatically adjusts the step size based on the local cell shape and vector field variation.

You can control the streamtrace integration by modifying the following parameters:

- **Step Size** - Specify the initial and maximum step size used while integrating through the vector field as a decimal fraction of the local cell or element width. A typical value (and the default) is 0.25, which results in four integration steps through each cell or element. The value for the Step Size affects the accuracy of the integration. Setting the Step Size too small can result in round-off errors, while setting it too large can result in truncation errors and missed cells.
 - **Max Steps** - Specify the maximum number of steps before the streamtrace is terminated. This prevents streamtraces from spinning forever in a vortex, or from wandering into a region where the vector components are very small, very random, or both. If you choose a small Step Size, you should specify a larger maximum number of steps.
 - **Minimum Step Size** - Specify the smallest step size for the Tecplot Engine to use. Setting this too small results in integration problems. Setting this greater than or equal to the Step Size results in a constant step size.
 - **Obey Source Blanking** - When active, streamtraces are generated for non-blanked regions only. When inactive, streamtraces are generated for blanked and unblanked regions.

Streamtraces may terminate at a zone boundary even if there is an adjacent zone into which the streamtraces should proceed. This can happen if there is a small gap between the zones. Specifying face neighbors in the data file to connect the zones can reduce this issue. Increasing the minimum integration step size can also eliminate this problem.



Example 34: Adding Volume Streamtraces

The following sample code demonstrates adding Volume Rod Streamtraces with a termination line to an existing 3D Plot:

```
StyleValue sv1;
sv1.set( TRUE, SV_STREAMTRACELAYERS, SV_SHOW);

TecUtilFieldLayerSetIsActive( SV_SHOWMESH, FALSE);

double xrawdata[] = { -.3203, -.0891 }; // x data for term line
double yrawdata[] = { .3829, -.4663 }; // y data for term line
TecUtilStreamtraceSetTermLine( 2, &(xrawdata[0]), &(yrawdata[0]));

TecUtilStreamtraceAdd( 1, Streamtrace_VolumeRod, StreamDir_Both,
                      0.5, 0.3311, 0.4310,
                      0.5, 0.3311, 0.4312);

StyleValue sv2(SV_STREAMATTRIBUTES);
sv2.set( TRUE, SV_TERMLINE, SV_ISACTIVE);
```



Example 35: Adding Streamtrace Markers

The following sample code illustrates adding streamtraces with multi-colored spherical markers to an existing 3D Plot:

```
StyleValue sv1;  
sv1.set( TRUE, SV_STREAMTRACELAYERS, SV_SHOW);  
TecUtilStreamtraceAdd( 1, Streamtrace_VolumeLine, StreamDir_Both,  
                      0.7, 0.8, 0.1,  
                      0.59, 0.59, 0.11);  
sv1.set( FALSE, SV_STREAMATTRIBUTES, SV_SHOWPATHS);  
  
TecUtilContourSetVariable(7);  
StyleValue sv2(SV_STREAMATTRIBUTES, SV_STREAMTIMING);  
sv2.set( TRUE, SV_SHOWMARKERS);
```

Iso-surfaces

An iso-surface is a surface that has a constant value for the contour variable. Iso-surfaces require that your data contains volume fieldmaps (IJK-ordered, finite-element brick, or finite-element tetrahedral fieldmaps). You can work with up to eight different iso-surface groups.

You may specify whether to include iso-surfaces on a fieldmap by fieldmap level.



If you opt to include iso-surfaces for a given fieldmap, Tecplot 360 will display all active iso-surface groups for that fieldmap.

You may specify the following attributes for a given iso-surface group:

- **Definition** - You may customize the following placement and definition settings for a given iso-surface group:
 - Whether to display the group
 - Which contour group to use
 - Whether to draw iso-surfaces at one, two or three specified value(s), or at each contour level. Default values are calculated by the Tecplot Engine.
- **Style** - You may customize the following style settings for a given iso-surface group:
 - **Mesh** - Whether to draw a mesh along the iso-surface, and customize the settings for the mesh.
 - **Contour** - Whether to draw contours along the iso-surface, and customize the settings for the contours.
 - **Shade** - Whether to display shading on the iso-surface, and customize the shade settings.



Example 36: Activating Iso-surfaces

The following example illustrates turning on iso-surfaces for an existing field plot and plotting one iso-surface defined at 3.57.

```
StyleValue sv;
sv.set(TRUE, SV_ISOSURFACELAYERS, SV_SHOW);
sv.set(3.57, 1, SV_ISOSURFACEATTRIBUTES, SV_ISOVALUE1);
```

Iso-surface Extraction



The code for this functionality is not output by the [Code Generator](#). You will need to use the code discussed in this section to enable the functionality.

Iso-surfaces are derived from the dataset and do not append the dataset. To extract existing iso-surfaces to Tecplot 360 zones and retain these surfaces after making changes to the contour variable, use `TecUtilCreateIsoZones`.

With this function, one zone is created for each iso-surface value displayed in the iso-surface. All of the variables in the dataset are interpolated from the 3D volume zones to the data points of the iso-surfaces.

Iso-surface zones are FE-surface quadrilateral element-type zones, regardless of the original 3D volume zone types. The mesh of the iso-surfaces is derived from the mesh of the original zones so that in regions where the original mesh was coarse, the iso-surface mesh is coarse, and where the original mesh was fine, the iso-surface mesh is fine.



After creating the new iso-surface zones, it is often a good idea to turn off or reconfigure the current settings for iso-surfaces because the new zones will occupy the same physical space as the original iso-surfaces. The new zones will not maintain the style, as defined by the derived iso-surfaces.

Slices

There are two types of slices:

- **Slices that are derived from the dataset** - Slices that are derived from the dataset are defined by a constant X, Y, or Z location or constant I, J, or K-index for IJK-ordered fieldmaps. This type of slice is part of the style of your plot and does not add to the dataset unless you extract it to a fieldmap.
- **Slices that are extracted directly to fieldmaps** - This option allows you to slice through 3D surface as well as 3D volume fieldmaps.

These operations are separate, and each has unique advantages. The resulting slices are always 3D surfaces.



You can use the [Code Generator](#) to create the code for these operations.



Example 37: Activating Slices

The following example illustrates turning on slices for an existing field plot:

```
StyleValue sv;  
sv.set(TRUE, SV_SLICELAYERS, SV_SHOW);
```

Slices that are derived from the dataset

You can add slices to your plot in order to view X, Y, or Z-planes within your data. With IJK-ordered data, slices can also be placed on I, J, or K planes. Up to eight different slice groups can be set. Each slice group can use different slice planes or different ranges for the same slice plane. Slices can include lighting effects, contours, meshes, and more.

For each slice group you can customize the following:

- **Slice Location** - Specify which plane the slice is drawn on (X,Y,Z, I, J, or K).
- **Slices to Display** - Specify whether to show the primary, start/end and/or intermediate slices.
- **Slice Attributes** - You can add and customize a contour layer, mesh layer, vector layer, edge layer, shade layer, and lighting for each slice group.

Slices that are extracted directly to fieldmaps

In most cases it is not necessary to extract slices to zones. However, there may be situations where you need to display multiple sets of slices in various directions. It is also possible to generate arbitrarily oriented slices when extracting to a zone.

To create zones out of slices derived from the dataset, call `TecUtilCreateSliceZones`. This option will create a separate zone for each slice plane. The created zones are FE quadrilateral, regardless of the source zone types. The style settings from the resulting zones can be set in the same fashion as the style settings for any other zone.

To create a slice zone defined by an arbitrary set of points, call `TecUtilCreateSliceZoneFromPlaneX`.



Example 38: Extracting an Arbitrary Slice

The following sample code demonstrates how to extract an arbitrary slice:

```
ArgList arglist;
arglist.appendDouble( SV_ORIGINX, 0.3);
arglist.appendDouble( SV_ORIGINY, 0.6);
arglist.appendDouble( SV_ORIGINZ, 0.5);
arglist.appendDouble( SV_NORMALX, 0.2);
arglist.appendDouble( SV_NORMALY, 0.75);
arglist.appendDouble( SV_NORMALZ, 0.3216);
TecUtilCreateSliceZoneFromPlaneX(arglist.getRef());
```

13 - 4 Plotting Subsets of Data



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

Once you have loaded your data, you can modify your field plot attributes. You may select the source (points or surfaces) for the data points used to plot vectors and scatter symbols.

13 - 4.1 Points

The points to plot can be specified for both fieldmaps and linemaps. You may select one of the following options to specify how the points are plotted:

- **Nodes on Surfaces** - Draws only the nodes that are on the surface of the fieldmap.
- **All Nodes** - Draws all nodes in the fieldmap.
- **All Connected** - Draws all the nodes that are connected by the node map. Nodes without any connectivity are not drawn.
- **Cell Centers Near Surfaces** - Draws points at the cell centers that are on or near the surface of the fieldmap.
- **All Cell Centers** - Draws points at all cell centers in the fieldmap.
- **Index Skip** - Specifies the skip intervals for the I, J, and K indices.



Example 39: Limiting the Points Plotted on a XY Line Plot

The following sample code demonstrates an XY Line Plot where data values are restricted to the J-range of 3-5:

```
Set objectSet(1);
StyleValue LineMapStyle(SV_LINEMAP);
LineMapStyle.set( 3, objectSet, SV_INDICES, SV_JRANGE, SV_MIN);
LineMapStyle.set( 5, objectSet, SV_INDICES, SV_JRANGE, SV_MAX);
```



Example 40: Plotting Nodes on the Surface(s) Only

The following sample code demonstrates how to modify an existing 3D Field Plot to show only nodes on the surfaces:

```
Set zone_set(1);
TecUtilZoneSetScatterSymbolShape( SV_GEOMSHAPE,
                                   zone_set.getRef(),
                                   GeomShape_Sphere);

StyleValue sv(SV_FIELDMAP);
sv.set( PointsToPlot_SurfaceNodes, zone_set, SV_POINTS,
        SV_POINTSTOPLOT);
```

13 - 4.2 Surfaces

There are many ways to divide volume data for plotting. One way to view volume data is to select surfaces from part of the data.

The following surface selections are available:

- **Boundary Cell Faces** - Plots all surfaces on the outside of the volume fieldmap.
- **IJK-ordered Data** - The minimum and maximum I, J, and K planes are plotted.
- **Finite-element Volume Data** - All faces that do not have a neighbor cell (according to the connectivity list) are plotted.

If blanking is turned on, the boundary cells in the blanked region will not be drawn and you will be able to see the interior of the volume fieldmap.

- **Exposed Cell Faces (default)** - When value blanking is used, the outer cell faces between blanked and non-blanked cells and the outer surfaces of the data are drawn.
- **Planes Settings (I, J, K, IJ, JK, IK, and IJK-planes)** - Plots the appropriate combination of I, J, and/or K planes. These settings are available only for IJK-ordered data.
- **Every Surface (Exhaustive)** - This setting will plot every face of every cell in volume data. It is not recommended for large datasets.



Example 41: Plotting Boundary Cell Faces Only

The following sample code demonstrates how to modify an existing 3D Field Plot to show Boundary Cell Faces with blanking activated:

```
TecUtilContourSetVariable(4);
TecUtilFieldLayerSetActive( SV_SHOWCONTOUR, TRUE);
TecUtilFieldLayerSetActive( SV_SHOWMESH, FALSE);

StyleValue sv1( SV_BLANKING, SV_DEPTH);
sv1.set( TRUE, SV_INCLUDE);
sv1.set( 30.0, SV_FROMFRONT);

Set objectSet(1);
StyleValue sv2(SV_FIELDMAP, SV_SURFACES, SV_SURFACESTOPLOT);
sv2.set( SurfacesToPlot_BoundaryFaces, objectSet);
```

I, J, and K-planes

A K-plane is the connected surface of all points with a constant K-index value. The I and J indices range over their entire domains, similarly for an I-plane and a J-plane. Examples of I, J, and K planes are shown below.



Example 42: Plotting Only the 7th J-plane

The following sample code illustrates how to display only the seventh J-plane in fieldmap 1:

```
Set objSet(1);
StyleValue sv( SV_FIELDMAP, SV_SURFACES);
sv.set( SurfacesToPlot_JPlanes, objSet, SV_SURFACESTOPLOT);
sv.set( 7, objSet, SV_IRANGE, SV_MIN);
sv.set( 7, objSet, SV_IRANGE, SV_MAX);
```

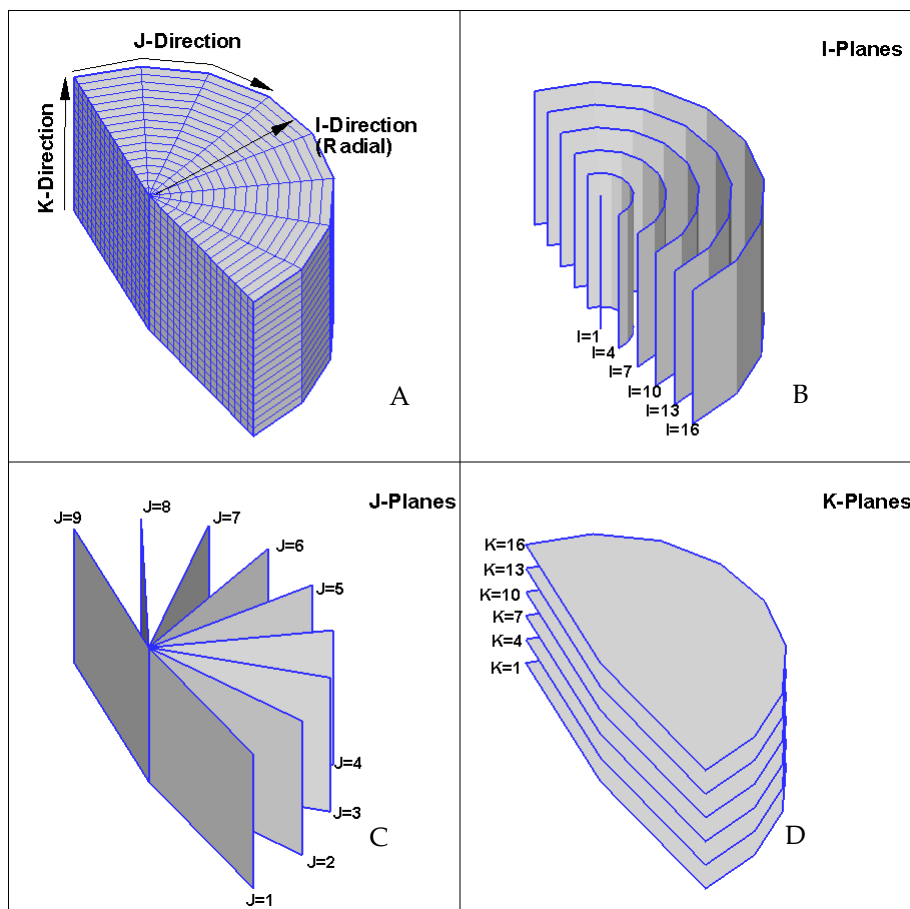


Figure 13-16. An illustration of IJK planes of a semi-circular zone (created by extracting a subzone from a circular zone). (A) All three planes with Surfaces to Plot set to “Boundary” via the Surfaces page of the Zone Style dialog. (B) Surfaces to Plot = I-planes (C) Surfaces to Plot = J-planes and (D) Surfaces



I, J, or K planes are not necessarily two-dimensional in physical space. They are called planes because they exist as planes in logical (IJK) space. In physical (XYZ) space, the planes may be cones, ellipsoids, or arbitrary surfaces.

13 - 5 Coloring



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

Each attribute of your plot can be set to a different color or color type. When assigning a value for `ColorIndex_t`, you have three types of color assignments:

- **Basic Colors** - Use the basic color palette to apply a single, constant color to a plot attribute (e.g. `Blue_C`, `Red_C`, etc.)

- [Multi-color Settings](#) - In many instances, you can assign a given plot attribute to be colored according by a contour group, and the coloring of the contour group is defined by the [Multi-color Settings](#). The Contour Variables are typically used for coloring mesh, contour, vector, and scatter layers.
- [RGB Multi-coloring](#) - RGB coloring is used to illustrate the relationship between two or three variables in your dataset, by setting R, G, and B to each of the variables.

For example, you can create a 3D field plot with a contour layer (with colors defined by a contour variable), an edge layer (with colors from the basic color palette), and a vector layer (with colors defined by RGB vectors).

13 - 5.1 Basic Colors

You can assign a given attribute to use a constant color value, by replacing `ColorIndex_t` from your [Code Generator](#) output to any of the following:

- `Black_C`
- `Blue_C`
- `Red_C`
- `Green_C`
- `Cyan_C`
- `Purple_C`
- `Yellow_C`
- `White_C`

Alternatively, you use a custom color by using `CustomNN_C`, where *NN* ranges from 1 to 64. You can design custom colors using the [Code Generator](#) and the `StyleValue` class.

13 - 5.2 Multi-color Settings



The contour variable must be defined prior to setting a color value to multi-coloring.

The colors used to display contour variables are determined by the global color map. By default, a color map called Small Rainbow is used, which is a rainbow of colors from blue to cyan to green to yellow to red. Color maps can also be linked to specific contour groups. Refer to [Section 13 - 3.1 "Fieldmap Layers"](#) for details.



The color map is used by all frames; if you change the color map to modify the look of one frame, all frames with contour flooding or any form of multi-coloring are modified as well.

To change the base color map, call `TecUtilColorMapSetBase`. You may select one of the following color maps:

- **Small Rainbow** - Five point color spectrum from blue to cyan to green to yellow to red.
- **Large Rainbow** - Seven point color spectrum from blue to cyan to green to yellow to red to purple to white.
- **Modern** - Seven point color spectrum; colors change in intensity from dark to light within each color band.
- **Gray Scale** - Color spectrum from black to white.
- **Wild** - Random color spectrum. Wild is different each time you select it.

- **Two-color** - A two-color spectrum.
- **User-defined** - A version of one of the first four options above that can be customized by the user. You can add or delete control points, as well as change RGB values for each control point.
- **Raw User-defined** - A version of one of the first four options above that can be customized by the user. NOTE: You can have only one raw user-defined color map at a time.

You can color an attribute using a given contour group (and therefore color map), by replacing `ColorIndex_t` from your [Code Generator](#) output with `MultiColor_C` or `MultiColor n _C`, where n can be 2 - 8. Refer to [Section "Contour"](#) on page 124 for additional information on working with contour groups.



Example 43: Changing the Global Colormap

The following sample code demonstrates changing global colormap number 1 to use the "modern" colormap:

```
TecUtilFieldLayerSetActive(SV_SHOWCONTOUR, TRUE);
StyleValue sv;
sv.set(ColorMap_Modern, 1, SV_GLOBALCOLORMAP, SV_CONTOURCOLORMAP);
```

13 - 5.3 RGB Multi-coloring

RGB multi-coloring occurs when Red, Green, and Blue values are supplied at each vertex. It may be used to create special flooding such as for Oil/Water/Gas or vector direction plots. RGB coloring may be used for each field plot object: zone layers, the mesh or contour layer for streamtraces or iso-surfaces, or any of the layers for slices. This affects multi-coloring for that object as well as any contour flooding. With RGB coloring, multi-colored objects such as vectors or scatter symbols have their color determined based on the RGB components of the field variables at their location. Multi-colored mesh and contour lines use the average value across the mesh line.



Exported Vector-based Files Limitation in RGB Coloring.

Vector-based export files such as WMF cannot show continuous RGB flooding. Objects that use RGB flooding are reduced to contain average cell flooding where each cell is flooded a solid color based on the averages of the RGB values at each vertex. The user is warned before such output is generated.

RGB Coloring is set by using the value `RGBColor_C` where `ColorIndex_t` is specified in your [Code Generator](#) output. NOTE: The RGB variables must be assigned before any object is set to be colored with RGB Coloring.

RGB Coloring Options

If your data has only two RGB variables, or if the sum of the variables is not normalized, you can adjust the settings using the following options:

- **RGB Mode** - You can either specify all three variables or specify two of the three variables and calculate the third. The third variable is calculated using the following formula $f(R)+f(G)+f(B)=1.0$ (assuming $f()$ is a function that maps R,G,B values into $[0,1.0]$).
- **Channel Variables** - Assign the variables that supply the values for the color components, as specified in the RGB Mode.
- **Channel Variable Range** - By default, it is assumed that the minimum value for any of the Channel Variables is zero, the maximum is one, and the sum of the three variables is one at every point. If the sum is not normalized, you can set a new minimum and maximum. For example, if your variables sum to 100 at every point, you can specify 100 for Value at Maximum Intensity.



Example 44: Creating a Vector Plot with RGB Coloring

The following sample code demonstrates a vector plot with RGB coloring. NOTE: This example assumes a 2D field plot has already been created.

```
TecUtilDataAlter( "{U}=1", NULL,
                  1, 0, 1,
                  1, 0, 1,
                  1, 0, 1,
                  FieldDataType_Float);

TecUtilDataAlter( "{V}=1", NULL,
                  1, 0, 1,
                  1, 0, 1,
                  1, 0, 1,
                  FieldDataType_Float);

TecUtilDataAlter( "{R}=X*X", NULL,
                  1, 0, 1,
                  1, 0, 1,
                  1, 0, 1,
                  FieldDataType_Float);

TecUtilDataAlter( "{G}=Y*Y", NULL,
                  1, 0, 1,
                  1, 0, 1,
                  1, 0, 1,
                  FieldDataType_Float);

TecUtilDataAlter( "{B}=X*Y", NULL,
                  1, 0, 1,
                  1, 0, 1,
                  1, 0, 1,
                  FieldDataType_Float);

StyleValue sv1(SV_GLOBA2WODVECTOR);
sv1.set( TecUtilVarGetNumByName("U"), SV_UVAR);
sv1.set( TecUtilVarGetNumByName("V"), SV_VVAR);

StyleValue sv2(SV_GLOBALRGB);
sv2.set(TecUtilVarGetNumByName("R"), SV_REDCHANNELVAR);
```

```
sv2.set(TecUtilVarGetNumByName("G"), SV_GREENCHANNELVAR);
sv2.set(TecUtilVarGetNumByName("B"), SV_BLUECHANNELVAR);

Set zoneSet(1);
StyleValue sv3(SV_FIELDMAP, SV_VECTOR);
sv3.set( RGBColor_C, zoneSet, SV_COLOR);

TecUtilFieldLayerSetActive( SV_SHOWMESH, FALSE);

// Must assign Vector Vars before this line.
TecUtilFieldLayerSetActive( SV_SHOWVECTOR, TRUE);
sv1.set(0.05, SV_RELATIVELENGTH);
```

RGB Legend

You can also include an RGB coloring legend in your plot. RGB legends have the following options:

- **Size and Position** - Specify the position of the anchor point as percentages of the frame width and height. (You can also move the legend interactively.) Specify the height of the legend in frame units.
- **Orientation** - Specify the order of the coloring channels (i.e. RGB, GBR, BGR etc.). The first channel listed is shown on the lower left corner, the second on the lower right, and the third at the top.
- **Text Labels** - You can opt to include text labels in your legend. If included, you can also customize the color and font of the text.
- **Red, Green, and Blue Label** - Each channel can be labeled by the name of the assigned variable, or by text you enter. When a channel has been calculated (no variable assigned), no label is shown.
- **Legend Box** - Specify which kind of box you want drawn around the legend (No Box, Filled, or Plain). If you choose Filled or Plain, format the box using the following controls:
 - **Line Thickness** - Specify the line thickness as a percentage of the frame height.
 - **Box Color** - Choose a color for the legend box outline.
 - **Fill Color (Filled only)** - Choose a color for the legend box fill.
 - **Margin** - Specify the margin between the legend text and legend box as a percentage of the text height.



Example 45: Including an RGB Legend

The following example code illustrates how to incorporate an RGB Legend in an existing field plot:

```
StyleValue sv1;
sv1.set( 3, 1, SV_GLOBALCONTOUR, SV_VAR);
sv1.set( TRUE, SV_FIELDLAYERS, SV_SHOWCONTOUR);

StyleValue sv2(SV_GLOBALRGB);
sv2.set( 1, SV_REDCHANNELVAR);
sv2.set( 2, SV_GREENCHANNELVAR);
```

```
sv2.set( 3, SV_BLUECHANNELVAR);
sv2.set( TRUE, SV_LEGEND, SV_SHOW);

Set objectSet(1);
sv1.set( ContourColoring_RGB, objectSet, SV_FIELDMAP,
        SV_CONTOUR, SV_FLOODCOLORING);
```

13 - 6 Axes



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

The axes for 3D, 2D, XY, and Polar plot types are automatically enabled. When generating the axes, axis labels and position, spacing, and labels for tick marks are also created.

The Tecplot Engine maintains five distinct sets of axes, one for each plot type. Each plot type has its own axis settings. This way you may set different axis styles for each plot type. The [Code Generator](#) can produce all the code required for changing axis styles. To change an axis style, you must be in the appropriate plot type for that axis, e.g. you must be in the XY Line plot type to make changes to the XY Line axis styles.

It is also important to note that the XY Line axis can support up to five X-axes and five Y-axes. When setting styles for an XY Line axis, you must supply an `offset1` parameter when using the `StyleValue` class. This offset tells the Tecplot Engine which X or Y axis should be changed. By default, all linemaps use the first X-axis and the first Y-axis.



Example 46: Changing the Linemap Axis Variable Assignments

The following example code illustrates setting the x-axis to use variable 5 and the y-axis to use variable 9 for the second linemap:

```
Set objectSet(2);
StyleValue sv(SV_LINEMAP);
sv.set( "Map 2", objectSet, SV_NAME);
sv.set( 1      , objectSet, SV_ASSIGN, SV_XAXISVAR);
sv.set( 2      , objectSet, SV_ASSIGN, SV_YAXISVAR);
```

NOTE: you may omit the `objectSet` parameter to change the value for all linemaps.

13 - 6.1 Axis Variable Assignment

For 2D axes, the first and second variables in the dataset are assigned to the X and Y-axes, respectively. For 3D axes, the first three variables in the dataset are assigned to the X, Y, and Z-axes, respectively. Use the [Code Generator](#) and the `StyleValue` class to change variable assignments for 2D and 3D axes.

For line plots, assigning axis variables is part of defining the mappings.

13 - 6.2 Axis Range

When changing Axis Ranges, it is important to understand that certain axis settings can affect the axis range. Settings such as “Preserve Length when Changing Range” and “Dependency” can alter the behavior when changing the axis range. Please keep the following in mind:

- **Dependent Axes** - When changing one axis, the range of the other axis (or axes) may be changed automatically to conform to the dependency setting.
- **Aspect Ratio** - When changing the aspect ratio of the frame (Sketch, XY, and 2D), the axis ranges will be adjusted such that they obey the Viewport Position settings.
- **X- and/or Y-axis Range Frame Linking** - Adjusting the range (or frame aspect ratio) in one frame will affect all other linked frames.
- **Preserve Length when Changing Range** - If this option is turned off, the Viewport Position is changed when adjusting the axis range.
- **Changing Axis Variables** - The range for an axis fits the value of the first variable assigned to that axis. If you deactivate the current layer and activate another, it may be necessary to reset the axis range.



Example 47: Changing the Axis Range

The following example code illustrates changing the minimum and maximum values to display on the first X-axis in a line plot:

```
StyleValue sv1( SV_XYLINEAXIS, SV_XDETAIL);  
sv1.set( -5.0, 1, SV_RANGEMIN);  
sv1.set( 5.0 , 1, SV_RANGEMAX);
```

13 - 6.3 Tick Marks

Each axis can be marked with tick marks, and those tick marks may or may not be labeled with either numbers or with custom text strings. You can customize the following tick mark attributes:

- **Tick Mark Placement** - Specify whether to place tick marks on the axis line or a region of the grid border.
- **Length and Thickness** - Tick mark length and thickness can be set independently for tick marks and minor tick marks.
- **Number of Minor Tick Marks** - Specify the number of minor tick marks to display in between major tick markers. (Use “0” to hide minor tick marks).
- **Direction** - Specify whether tick marks should be drawn in, out, or towards the center.
- **Spacing** - Use auto-spacing or a manual spacing increment.
- **Labels** - Specify whether to display tick mark labels. You can customize the text and style formatting of your tick mark labels.

13 - 6.4 Axis Lines

The actual axis line is shown by default whenever the axis is shown. However, you can hide the axis line without turning off the axis as a whole. When an axis line is displayed, you can customize its placement, color and thickness.

13 - 6.5 Axis Labels

Axis labels are shown by default whenever the axis is shown. However, you can hide the axis labels without turning off the axis as a whole. When labels are displayed, you can customize their placement, color, thickness, and you can also create custom labels.

You can also display your labels in Time/Date format by using Tecplot 360 supports Excel¹ Time/Date number formats, with the exception of AM/PM time specifications, and long day and month names. This support allows you to create number formats in Excel, and then import them for use with your Tecplot 360 plots.

13 - 6.6 Axis Grid Area

The grid area of your plot is the area defined by the axes. For Sketch, XY Line, and 2D Cartesian plots you can alter the size of the grid area by changing the extents of the viewport. (For these plot types, the viewport and grid area are synonymous.) For Polar Line and 3D Cartesian plots, the grid area is altered by changes to the axis ranges. For 3D axes, you can also specify an axis box padding, the minimum distance from the data to the axis box, and whether to light-source shade the axis planes.

You can also control whether the grid area or viewport are color-filled.

13 - 7 Annotations

You can enhance any plot or create a drawing from scratch using the text and drawing tools. Tecplot 360 provides tools for creating polylines, circles, ellipses, squares, rectangles, and text. You can also insert BMP, JPEG, or PNG images to enhance your plot. You can further annotate your plots using labels and legends.

13 - 7.1 Text



The Text options provided by the Tecplot Engine are not accessible via the [Code Generator](#) and the StyleValue class. You will need to use the [along with the following section](#) to enable Text annotations.

In order to add text to the current frame, you must first call `TecUtilTextCreate` or `TecUtilText3DCreate`. This will allow you to specify the position, height and text string. Both functions return `Text_ID`, which is used to customize the text (i.e. set font, color, text box properties, etc.) You can change the value of the text string later with the `TecUtilTextSetString` function.

You can customize the following attributes of the text string:

- **Color** - Use `TecUtilTextSetColor` to set the color value for your text string. Refer to [Section 13 - 5 "Coloring"](#) for details on the various color options.
- **Font** - You can customize the text font style by either setting the font for the entire string with `TecUtilTextSetFont`, or you can embed text formatting tags in your text string to set the font style for a portion of your text string.
- **Text Formatting Tags** - The text formatting tags and their effects are as follows (format tags are not case sensitive and may be either upper or lower case):

1. Tecplot 360 does not currently support Mac Excel "1904" format.

- **Font Weight** - Use the bold or italics tags to customize the weight of your text string.
 - `...` - Bold
 - `<i>...</i>` - Italics
- **Verbatim** - Use the verbatim tag, `<verbatim>...</verbatim>`, to display a portion of your text string “as-is”, without any formatting.
- **Font Override** - Use any of the following tags to override the font settings set with `TecUtilTextSetFont` for part of the text string.
 - `<greek>...</greek>` - Greek Font
 - `$...$` - Math Font
 - `<userdef>...</userdef>` - User-defined Font
 - `<helvetica>...</helvetica>` - Helvetica Font
 - `<times>...</times>` - Times Font
 - `<courier>...</courier>` - Courier Font



The text formatting tags are not available when the font for the string is set to **Greek**, **Math**, or **User-defined**.

- **Subscripts and Superscripts** - You can produce subscripts or superscripts by enclosing any characters with `_{...}` or `^{...}`, respectively. Tecplot 360 has only one level of superscripts and subscripts; expressions requiring additional levels, such as e^{x^2} , must be created by hand using multiple text strings. If you alternate subscripts and superscripts, the superscript will be positioned directly above the subscript. Thus, the string `a_b^c` produces a_b^c . To produce consecutive superscripts, enclose all superscript characters in a single pair of tags. The string `x^(a+b)` produces $x^{(a+b)}$ in your plot.

To insert a tag into text literally, precede the first angle bracket with a backslash (“\”). To insert a backslash in the text, just type two backslashes (“\\”). In ASCII input files, the number of backslashes must be doubled (two to precede a special character, four to create a backslash) because the Preplot program also requires a backslash to escape special characters.

See also: [“Special Characters”](#) on page 156 and [“Dynamic Text”](#) on page 157 for additional special values that you can set using your text string.

- **Angle (deg)** - Specify the orientation of the text relative to the axis with `TecUtilTextSetAngle`. The angle is measured in degrees counter-clockwise from horizontal. Horizontal text is at 0 degrees; vertical text is at 90 degrees.
- **Height** - Specify the height for the text with `TecUtilTextSetHeight`. The units used for the height value are determined via `TecUtilTextSetCoordSysAndUnits`.
- **Coordinate System/Character Height** - Specify a combination of coordinate system and character height units via `TecUtilTextSetCoordSysAndUnits`. You can set the following combinations of coordinate system and height units:
 - **CoordSys_Frame/Units_Frame** - Specify character height as a percentage of frame height and place the text in a frame coordinate system.
 - **CoordSys_Frame/Units_Point** - Specify character height in points and place the text in a frame coordinate system.
 - **CoordSys_Grid/Units_Grid** - Specify character height in grid units, and place the text in the grid coordinate system.
 - **CoordSys_Grid/Units_Frame** - Specify character height in frame units and place it in the grid coordinate system.

- **Origin** - Use `TecUtilTextSetAnchorPos` to specify the X- and Y-coordinates of the text anchor. The text anchor sets the justification of the text within the text box. The text is anchored using the coordinate system specified in `TecUtilTextSetCoordSysAndUnits`. The anchor point of the text box is set with `TecUtilTextBoxSetAnchor`.
- **Clipping** - Clipping refers to displaying only that portion of an object that falls within a specified clipping region of the plot. If you have specified your text position in the **CoordSys_Frame** coordinate system, the text will be clipped to the frame. If you have specified the **CoordSys_Grid** coordinate system, you can choose to clip your text to the frame or the viewport by calling `TecUtilTextSetClipping`. The size of the viewport depends on the plot type as follows:
 - **3D Cartesian** - The viewport is the same as the frame, so viewport clipping is the same as frame clipping.
 - **2D Cartesian/XY Line** - The viewport is defined by the extents of the X and Y axes.
 - **Polar Line/Sketch** - By default, the viewport is the same as the frame.
- **Line Spacing** - Call `TecUtilTextSetLineSpacing` to set the line spacing for the text. The units used are specified with `TecUtilTextSetCoordSysAndUnits`.
- **Margin** - Specify the space between the text and box surrounding the text object as a percentage of the text character height via `TecUtilTextBoxSetMargin`.
- **Text Box** - The text string is housed within a text box (which you choose whether or not to display). Use `TecUtilTextBoxSetType` to set the style of the box around the text. Select from the following types:
 - **TextBox_None** - Specify that no box is drawn around the text.
 - **TextBox_Filled** - Use a filled box around the text. NOTE: A filled box is opaque; if you place it over another object, the underlying object cannot be seen.
 - **TextBox_Plain** - Use a plain box around the text.

If you select `TextBox_Plain` or `TextBox_Filled`, you can set the Line Thickness and line color with `TecUtilTextBoxSetLineThickness` and `TecUtilTextBoxSetColor`, respectively. If you select `TextBox_Filled`, you can set the box outline fill color with `TecUtilTextBoxSetFillColor`. Refer to [Section 13 - 5 "Coloring"](#) for information regarding the possible color values.

- **Text Box Anchor Location** - Specify the anchor point, or fixed point, for the text box with `TecUtilTextBoxSetAnchor`. As the text box grows or shrinks, the anchor location is fixed, while the rest of the box adjusts to accommodate the new size. There are nine possible anchor points, corresponding to the left, right, and center positions on the headline, midline, and baseline of the text box.



Example 48: Adding a Text Label

The following sample code generates a text label with Courier font and a centered anchor point:

```
Text_ID tid;
tid = TecUtilTextCreate( CoordSys_Grid, 0.0, 0.5, Units_Frame,
                        2.4, "Data Point 1");
TecUtilTextSetFont( tid, Font_Courier);
TecUtilTextSetAnchor( tid, TextAnchor_MidCenter);
```

Special Characters

Tecplot 360 supports the ISO-Latin one-character encodings. Characters in the ASCII ordinal range from 160-255 are available. [Table 13 - 1](#) shows the characters supported by the Tecplot 360. Note that the two right-hand columns represent the extended European characters. Text formatting tags for Greek, Math, or User-defined characters work only with characters in the range 32-126 and are not available for the extended European characters.

If your keyboard is configured to produce European characters, then the European characters should appear and print automatically with no additional setup.

If your keyboard is not configured to produce a specific European character, you can generate it by including the sequence `\nnn` in your text where `nnn` is from the character index table found in [Table 13 - 1](#). For example, if your keyboard will not generate the `é` and you want to show the word “latté,” use:

```
latt\233
```

In order to use the custom characters, you can either set the font type for the entire text string to Greek, Math, or User-defined (by using `TecUtilTextSetFont`), or use the appropriate text formatting tags around the desired characters.

Similarly, if you are using Greek, Math, or User-defined font type, you can use one of the “English” formatting tags around the subset of characters to display the English characters.

Character Index	English Text	Greek	Math	User Defined	Character Index	English Text	Greek	Math	User Defined	Character Index	Extended Character	Character Index	Extended Character
032	(space)				080	P	Π	∠	∞	160	ı	208	Đ
033	!	!	Y		081	Q	Θ	∇	◊	161	ı	209	Ñ
034	*	*	√		082	R	Ρ	⊗	◊	162	ı	210	Ô
035	#	#	≤		083	S	Σ	⊗	◊	163	ı	211	Ó
036	\$	\$	/		084	T	Τ	™	◊	164	ı	212	Ö
037	%	%	∞		085	U	Υ	Π	◊	165	ı	213	Õ
038	&	&	f		086	V	ς	√	⊗	166	ı	214	Ö
039	'	'	+		087	W	Ω	·	•	167	ı	215	×
040	((+		088	X	Ξ	∩	•	168	ı	216	Ø
041))	♥		089	Y	Ψ	∧	•	169	ı	217	Ú
042	*	*	▲		090	Z	Ζ	∇	•	170	ı	218	Ú
043	+	+	↔		091	[[↔		171	ı	219	Ú
044	,	,	↑		092	\	∴	↔		172	ı	220	Ú
045	-	-	↑		093]]	↔		173	ı	221	Ý
046	.	.	→		094	^	⊥	⇒		174	ı	222	Þ
047	/	/	↓		095	'	—	↓		175	ı	223	ß
048	0	0	°		096	—	—	◊		176	ı	224	à
049	1	1	±		097	a	α	◊	•	177	ı	225	á
050	2	2	″		098	b	β	⊗	•	178	ı	226	â
051	3	3	≥		099	c	χ	⊗	•	179	ı	227	ã
052	4	4	×		100	d	δ	™	•	180	ı	228	ä
053	5	5	∞		101	e	ε	Σ	•	181	ı	229	å
054	6	6	∂		102	f	φ	∫	◊	182	ı	230	æ
055	7	7	•		103	g	γ	∫	◊	183	ı	231	ç
056	8	8	+		104	h	η	∫		184	ı	232	è
057	9	9	≠		105	i	ι	∫		185	ı	233	é
058	:	:	≡		106	j	φ	∫		186	ı	234	ê
059	;	;	≈		107	k	κ	∫		187	ı	235	ë
060	<	<	∫		108	l	λ	∫		188	ı	236	ì
061	=	=	∫		109	m	μ	∫		189	ı	237	í
062	>	>	∫		110	n	ν	∫		190	ı	238	î
063	?	?	∫		111	o	ο	∫		191	ı	239	ï
064	⊗	⊗	∫		112	p	π	∫	•	192	ı	240	ò
065	A	A	∫		113	q	θ	∫		193	ı	241	ñ
066	B	B	∫	+	114	r	ρ	∫		194	ı	242	ò
067	C	X	∫	x	115	s	σ	∫		195	ı	243	ó
068	D	Δ	∫	*	116	t	τ	∫		196	ı	244	ô
069	E	E	∫	Δ	117	u	υ	∫		197	ı	245	õ
070	F	Φ	∫	∇	118	v	ϖ	∫		198	ı	246	ö
071	G	Γ	∫	□	119	w	ω	∫		199	ı	247	÷
072	H	H	∫	◊	120	x	ξ	∫		200	ı	248	ø
073	I	I	∫	◊	121	y	ψ	∫		201	ı	249	ù
074	J	θ	∫	◊	122	z	ζ	∫		202	ı	250	ú
075	K	K	∫	*	123	{	{	∫		203	ı	251	û
076	L	Λ	∫	*	124			∫		204	ı	252	ü
077	M	M	∫	+	125	}	}	∫		205	ı	253	ý
078	N	N	∫	◊	126	-	-	∫		206	ı	254	þ
079	O	O	∫	◊	127					207	ı	255	ÿ

Table 13 - 1: Character Indices

Dynamic Text

You can add dynamic text strings to your text using the syntax below. When dynamic text strings are used, the displayed text is updated as the data changes.

Variables	Notes
&(AUXDATASET:name)	The value of the named auxiliary data attached to the dataset.
&(AUXFRAME:name)	The value of the named auxiliary data attached to the frame.
&(AUXPAGE:name)	The value of the named auxiliary data attached to the page.
&(AUXVAR[nnn]:name)	The value of the named auxiliary data attached to variable <i>nnn</i> .

Variables	Notes
&(AUXLINEMAP[Q]:name)	The value of the named auxiliary data attached to linemap Q, where Q = either <i>nnn</i> or <i>ACTIVEOFFSET = nnn</i> and <i>nnn</i> = linemap number. If <i>ACTIVEOFFSET=</i> is used, the integer value indicates the first linemap associated with the <i>nnn</i> th active fieldmap.
&(AUXZONE[Q]:name)	The value of the named auxiliary data attached to Q, where Q = either <i>nnn</i> or <i>ACTIVEOFFSET = nnn</i> and <i>nnn</i> = zone number. If <i>ACTIVEOFFSET=</i> is used, the integer value indicates the first zone associated with the <i>nnn</i> th active fieldmap.
&(AXISMAX n)	Maximum value of the current n -axis range, where n is one of: A ^a , R, X, Y, or Z.
&(AXISMIN n)	Minimum value of the current n -axis range, where n is one of: A ^a , R, X, Y, or Z.
&(BYTEORDERING)	Displays the platform's byte ordering (INTEL or MOTOROLA).
&(DATE)	The current date, in the format <i>dd Mmm yyyy</i> .
&(DATASETFNAME[<i>nnn</i>])	Filename of the <i>nnn</i> th file associated with the current dataset. If <i>nnn</i> is omitted, then all dataset filenames are shown, separated by new lines.
&(DATASETTITLE)	The current dataset title.
&(ENDSLICEPOS[< <i>slice group</i> or <i>activeoffset</i> >])	The position of the ending slice plane.
&(EXPORTISRECORDING)	Returns "YES" if recording is active, otherwise returns "NO".
&(FRAMEName)	The frame name.
&(INBATCHMODE)	Returns a value of 1 if the software is in batch mode, 0 if interactive.
&(ISDATASETAVAILABLE)	Returns a value of 1 if a dataset exists for the current frame, 0 if nonexistent.
&(ISOSURFACELEVEL[< <i>iso surface group</i> or <i>activeoffset</i> >] [<i>nnn</i>])	The value of the contour variable on the <i>nnn</i> th iso-surface.
&(LAYOUTFNAME)	The name of the current layout file.
&(LOOP)	Innermost loop counter.
&(MACROFILEPATH)	Path to the folder containing the most recently opened macro file.
&(MAX n)	Maximum value of the n variable, where n is one of: A ^a , R, X, Y, or Z. For 2D or 3D Cartesian plots, the value is calculated from all active zones. For line plots, the value is calculated from the zone assigned to the first active linemap.
&(MAXB)	Maximum value of the blanking variable for the first active constraint. For 2D or 3D Cartesian plots, the value is calculated from the active zones. For line plots, the value is calculated from the zone assigned to the first active linemap.
&(MAXC)	Maximum value of the contour variable for contour group 1. For 2D or 3D Cartesian plots, the value is calculated from the active zones. For line plots, the value is calculated from the zone assigned to the first active linemap.
&(MAXI), &(MAXJ), &(MAXK)	[I, J, K]-dimension of the first active zone for 2D and 3D Cartesian plot types. For finite-element data, I represents the number of nodes in the first active zone, J represents the number of elements in the first active zone, and K represents the number of nodes per element (cell-based) or total number of faces (face-based) in the first active zone.

Variables	Notes
&(MAXS)	Maximum value of the scatter sizing variable of the active zones.
&(MAXU), &(MAXV), &(MAXW)	Maximum value of the variable assigned to the [X, Y, Z]-vector component of the active zones.
&(MAXVAR[<i>nnn</i>])	Maximum value of variable <i>nnn</i> .
&(MIN n)	Minimum value of the n variable, where n is one of: A ^a , R, X, Y, or Z. For 2D or 3D Cartesian plots, the value is calculated from all active zones. For line plots, the value is calculated from the zone assigned to the first active linemap.
&(MINB)	Minimum value of the blanking variable of the first active blanking constraint. For 2D or 3D Cartesian plots, the value is calculated from all active zones. For line plots, the value is calculated from the zone assigned to the first active linemap.
&(MINC)	Minimum value of the contour variable of contour group 1. For 2D or 3D Cartesian plots, the value is calculated from all active zones. For line plots, the value is calculated from the zone assigned to the first active linemap.
&(MINS)	Minimum value of the scatter sizing variable for the active zones.
&(MINU), &(MINV), &(MINW)	Minimum value of the variable assigned to the [X, Y, Z]-vector component for the active zones.
&(MINVAR[<i>nnn</i>])	Minimum value of variable <i>nnn</i> .
&(NUMFRAMES)	Number of frames.
&(NUMPROCESSORS USED)	Number of processors used. This may be different than the total number on the machine because of the \$!Limits MaxAvailableProcessors configuration file command, or because of a product limitation. Tecplot Focus is limited to one processor, while Tecplot 360 is limited to eight.
&(NUMVARS)	Number of variables in the current dataset.
&(NUMXYMAPS)	Number of XY-linemap assigned to the current frame.
&(NUMZONES)	Number of zones in current dataset.
&(OPSYS)	Displays the current operating system. 1=UNIX/Linux/Macintosh, 2=Windows.
&(PAPERHEIGHT)	The paper height (in inches).
&(PAPERWIDTH)	The paper width (in inches).
&(PLATFORM)	The platform type (e.g. SGI or WINDOWS).
&(PLOTTYPE)	Plot type of the current frame: 0 for Sketch, 1 for XY Line, 2 for Cartesian 2D, 3 for Cartesian 3D, and 4 for Polar Line.
&(PRIMARYSLICEPOS [<slice group active offset>])	The primary slice position.
&(PRINTFNAME)	The name of the current print file.
&(SLICEPLANETYPE[<slice group active offset>])	The type of the slice plane (X, Y, Z, I, J or K-planes).
&(SOLUTIONTIME)	The current solution time.

Variables	Notes
&(SOLUTIONTIME[Q])	Solution time of Q, where Q = either <i>nnn</i> or <i>ACTIVEOFFSET</i> = <i>nnn</i> and <i>nnn</i> = zone number. If <i>ACTIVEOFFSET</i> = is used, the integer value indicates the first zone associated with the <i>nnn</i> th active fieldmap. &(SOLUTIONTIME[5]) displays the solution time of the 5 th zone. &(SOLUTIONTIME[ACTIVEOFFSET=3]) displays the solution time of the first zone in the 3 rd active fieldmap.
&(STARTSLICEPOS[< <i>slice group</i> or <i>activeoffset</i> >])	The position of the starting slice plane.
&(STRANDID[<i>x</i>])	The strandID of a zone in dynamic text.
&(STREAMSTARTPOS[<i>nnn</i>])	Starting position (X, Y, Z) of the <i>nnn</i> th streamtrace.
&(STREAMTYPE[<i>nnn</i>])	Type (Surface Line, Volume Line, Volume Ribbon, Volume Rod) of the <i>nnn</i> th streamtrace.
&(\$ <i>string</i>)	The value of the system environment variable <i>string</i> .
&(TECHOME)	Path to the home directory.
&(TECPLOTVERSION)	Displays the version number.
&(TIME)	The current time, in the format <i>hh:mm:ss</i> .
&(VARNAME[<i>nnn</i>])	The variable name of variable <i>nnn</i> .
&(ZONEMESHCOLOR[Q])	Color of the mesh for Q, where Q = either <i>nnn</i> or <i>ACTIVEOFFSET</i> = <i>nnn</i> and <i>nnn</i> = zone number. If <i>ACTIVEOFFSET</i> = is used, the integer value indicates the <i>nnn</i> th active zone for field plots or the zone associated with the <i>nnn</i> th active linemap for line plots.
&(ZONENAME[Q])	The zone name of Q, where Q = either <i>nnn</i> or <i>ACTIVEOFFSET</i> = <i>nnn</i> and <i>nnn</i> = zone number. If <i>ACTIVEOFFSET</i> = is used, the integer value indicates the <i>nnn</i> th active zone for field plots or the zone associated with the <i>nnn</i> th active linemap for line plots.

a. where A represents the theta (or angle) axis variable in Polar Line plots.

The placeholders must be typed exactly as shown, except that the *nnn* in the zone name and variable name placeholders should be replaced by the actual number of the zone or variable, such as &(ZONENAME[3]) or &(VARNAME[2]).

You can, of course, embed the dynamic text strings in text records in a Tecplot-format data file, as in the following example:



Example 49: Adding Dynamic Text to your Plot

The following example code illustrates including a text label that displays the date in your plot:

```
TecUtilTextCreate( CoordSys_Frame, 10.0, 10.0,
                  Units_Point, 30.0, "&(DATE)");
```

System environment variables can be accessed directly from the Tecplot 360 by using the following: `&($string)`, where `string` is the name of your environment variable. Using environment variables within Tecplot 360 can add another degree of flexibility by taking advantage of your customized environment. If an environment variable is missing, the environment variable name itself will appear on the screen.

Formatting Dynamic Text Strings

If you want a dynamic text string to be formatted in a specific way, you can include C-style number formatting strings in the macro variable specification.

The syntax for including a format string is:

```
&(DynamicTextString%formatstring)
```

The following formats are available:

- **s** - string of characters
- **d** - signed integer
- **e** - scientific notation with a lowercase "e"
- **E** - scientific notation with an uppercase "E"
- **f** - floating point
- **g** - use %e or %f, whichever is shorter
- **G** - use %E or %f, whichever is shorter
- **u** - unsigned integer, written out in decimal format
- **o** - unsigned integer, written out in octal format
- **x** - unsigned integer, written out in hexadecimal (where a - f are lowercase)
- **X** - unsigned integer, written out in hexadecimal (where A - F are uppercase)

For example, to display the message "Maximum contour value is: `xxxxxx`" where `xxxxxx` only has two digits to the right of the decimal place. You would use:

```
"Maximum contour value is: &(MAXC%.2f)"
```

If `|MAXC|` currently has a value of 356.84206 then:

```
"Maximum contour value is: 356.84"
```

If, in the above example, you wanted to use exponential format you could use:

```
"Maximum contour value is: &(MAXC%12.6e)"
```

Here the result would be:

```
"Maximum contour value is: 3.568421e+02"
```

13 - 7.2 Geometries




Geometries in the Tecplot Engine are simply line drawings. Geometries include polylines (a set of line segments), circles, ellipses, rectangles, and squares.



The Geometry options provided by the Tecplot Engine are not accessible via the [Code Generator](#) and the StyleValue class. You will need to use the [Code Generator](#) along with the following section to enable Geometries.

- **Line** - There are number of different types of lines that you can create with the Tecplot Engine. You can create single line segments (i.e. a single line with two defined sets of coordinates), polylines (i.e. a single line with multiple coordinate pairs), or multi polylines (i.e. multiple lines, each with multiple coordinate pairs).
 - **Line Segments** - A line segment has two defined endpoints. Call `TecUtilGeom2DLineSegmentCreate` or `TecUtilGeom3DLineSegmentCreate` to add a line segment to your plot.
 - **Polylines** - A polyline is a line created by a group of line segments. You can add a polyline to your plot with `TecUtilGeom2DPolylineCreate` or `TecUtilGeom3DPolylineCreate`.
 - **Multiple Polylines** - You can create multiple polylines simultaneously by calling `TecUtilGeom2DMPolyCreate` or `TecUtilGeom3DMPolyCreate`. After calling the appropriate *create* function, you will also need to call `TecUtilGeom2DMPolySetPolyline` or `TecUtilGeom3DMPolySetPolyline` to define the coordinates of the points in each polyline.

Once the line(s) have been created, you can customize the arrowheads of the line using any of following options:

- **Attachment** - Use `TecUtilGeomArrowheadSetAttach` to specify whether the arrowheads should be attached at: the beginning of the segment, the end of the segment, both ends, or not at all.
- **Size(%)** - Use `TecUtilGeomArrowheadSetSize` to specify the size of the arrowhead, as a percentage of frame height.
- **Style** - Use `TecUtilGeomArrowheadSetStyle` to specify the style for your arrowhead. The following styles are available:
 -  Plain arrowhead style
 -  Filled arrowhead style
 -  Hollow arrowhead style
- **Angle** - Use `TecUtilGeomArrowheadSetAngle` to specify the angle the arrowhead makes with the polyline.
- **Circle** - Use `TecUtilGeomCircleCreate` to add a circle to your plot. To set the number of points used to draw the circle use `TecUtilGeomEllipseSetNumPoints`.
- **Ellipse** - Use `TecUtilGeomEllipseCreate` to add an ellipse to your plot. Use `TecUtilGeomEllipseSetNumPoints` to set the number of points used to draw the circle.
- **Square** - Use `TecUtilGeomSquareCreate` to add a square to your plot.
- **Rectangle** - Use `TecUtilGeomRectangleCreate` to add a rectangle to your plot.

Geometry Style

When you create a given geometry with the Tecplot Engine, the create function returns `Geom_ID`. You can then use `Geom_ID` to customize the style and display attributes of that geometry.

The following options are available:

- **Line Color** - Use `TecUtilGeomSetColor` to set the line or outline color of the specified geometry. Refer to [Section 13 - 5 "Coloring"](#) for details regarding color options. NOTE: Multi-Coloring and RGB Coloring are not available for geometries.
- **Line Pattern** - Use `TecUtilGeomSetLinePattern` to specify the desired pattern (Solid, Dashed, Dotted, LongDash, or DashDotDot).
- **Pattern Length (%)** - Use `TecUtilGeomSetPatternLength` to specify the length of the line pattern as a percentage of the frame width.
- **Line Thickness (%)** - Use `TecUtilGeomSetLineThickness` to specify the thickness of the line as a percentage of the frame width.
- **Fill Color** - Use `TecUtilGeomSetFillColor` to set the line or outline color of the specified geometry. Refer to [Section 13 - 5 "Coloring"](#) for details regarding color options. NOTE: Multi-Coloring and RGB Coloring are not available for geometries.
- **Origin** - Use `TecUtilGeomSetAnchorPos` to set the X and Y-coordinates of the anchor position of the geometry. For circles and ellipses, the anchor is the center. For squares and rectangles, it is the base corner. For lines, it is the offset added to all points of the geometry. The units used are set via `TecUtilGeomSetPosCoords`.
- **Coordinate System** - Specify the coordinate system for the geometry (Frame or Grid) via `TecUtilGeomSetPositionCoordSys`.
 - **CoordSys_Frame** - The geometry is always displayed at constant size when you zoom in or out of the plot.
 - **CoordSys_Grid** - The geometry resizes with the data grid.
- **Clipping** - Clipping refers to displaying only that portion of an object that falls within a specified clipping region of the plot. You can specify which form of clipping to use via `TecUtilGeomSetClipping`.

If you have specified your geometry position in the Frame coordinate system, the geometry will be clipped to the frame—any portion of the geometry that falls outside the frame is not displayed. If you have specified the Grid coordinate system, you can choose to clip your geometry to the frame or the viewport. The size of the viewport depends on the plot type as follows:

- **3D Cartesian** - The viewport is the same as the frame, so viewport clipping is the same as frame clipping.
- **2D Cartesian/XY Line** - The viewport is defined by the extents of the X and Y axes.
- **Polar Line/Sketch** - By default, the viewport is the same as the frame.
- **Draw Order** - Geometries can be drawn either before the data or after the data via `TecUtilSetDrawOrder`. If a geometry is drawn before the data, the plot layers, such as mesh, contour lines, etc. will be drawn on top of the geometry. If a geometry is drawn after the data, the geometry will be drawn last, obscuring the data.



You can place text and geometries in any order you like. The Tecplot Engine draws all geometries first, in the order in which they were placed, then all text. Use the `TecUtilFramePopXXX` and `TecUtilFramePushXXX` commands to reorder objects in the viewstack.



Example 50: Adding a Square Geometry to your Plot

The following sample code creates a green square with a black outline:

```
Geom_ID gid = TecUtilGeomSquareCreate( CoordSys_Frame,
                                         20.0, 20.0,
                                         60.0);

if ( gid != TECUTILBADID )
{
    TecUtilGeomSetLineThickness( gid, 0.6);
    TecUtilGeomSetColor(        gid, Black_C);
    TecUtilGeomSetFillColor(    gid, Green_C);
    TecUtilGeomSetIsFilled(     gid, TRUE);
}
```

13 - 7.3 Images



The Image options provided by the Tecplot Engine are not accessible via the [Code Generator](#) and the StyleValue class. You will need to use the [Code Generator](#) section to enable Images.

You can import images from JPEG, BMP, and PNG files by calling `TecUtilGeomImageCreate`. These images can be used as logos or as a backdrop to your plot. Images cannot be included in data files. When you save a data file, even if you specify to include geometries, any images in the plot are not saved.

In layout and style sheet files, the image is referenced from its original location. This reference can be a relative reference or an absolute (as with data files).

After you have imported the image, you can customize the following attributes:

- **Height and Width** - Call `TecUtilGeomImageSetHeight` or `TecUtilGeomImageSetWidth` to set the height or width of the image, respectively. The image will be resized to fit the dimension specified. As such, the size of a geometry when it is displayed in a frame is not necessarily the size of the image in the file.
- **Resize Filter** - The Resize Filter determines how the image is resized to fit the screen. The following filters are available:
 - **Fast (textures)** - *default*- OpenGL textures are used to resize the image. This is the fastest option (given sufficient graphics space). However, the accuracy of the image may suffer, especially when reducing an image to a size much smaller than it was before.
 - **Pixelated** - Choose this option when the image is much larger than its original size and you want to see the individual pixels. This option is slower than the Fast (textures) filter for increasing the size of images.

- **Smooth** - There are seven smoothing options, all producing slightly different effects. These options are slower than the Fast (textures) filter, but produce better effects for highly reduced images. In general, use the Smooth (Lanczos2) option unless you have specific image processing needs.



The resize filter has no effect on vector-based output, only on the screen and for exported images.

13 - 7.4 Labels

You can label all or some of the data points or nodes in your field plots with either the index value(s) of the data point or the value of some specified variable at each point. You can also label each cell or element of the data with its index (which for finite-element data is its element number).

You can label all or some of the data points or nodes in your line plots with either the index of the data point, the value of the dependent variable at the point, or both the values (X & Y or Theta & R) for the data point.

Use the [Code Generator](#) to determine the syntax for including labels in your plots.

13 - 8 Blanking



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

Blanking allows you to exclude specific portions of fieldmaps from being plotted (in other words, selectively display certain cells or data points). In 3D, the result is analogous to a cutaway view. In general, all types of blanking affect all field layers, fieldmaps, and all other plot attributes with the following exceptions:

Plot attributes not affected by blanking:

Type of Blanking	Attribute(s) Excluded from Blanking
Value Blanking	Edge Layer
IJK Blanking	Derived Objects (Slices, Streamtraces, or Iso-surfaces) FE Zones Unorganized Zones
Depth Blanking	Derived Objects (Slices, Streamtraces, or Iso-surfaces)

Table 13 - 2: Plot attributes not affected by blanking.

Blanking settings are only applied to the current frame. Value blanking settings for multiple frames may be synchronized using frame linking. Use the [Code Generator](#) to determine the syntax for your blanking needs.

In the following discussions, the term “cell” is used. In I-ordered datasets, a cell is the connection between two adjacent points. In IJ-ordered datasets, a cell is the quadrilateral area bounded by four neighboring data points. In IJK-ordered datasets, a cell is the six-faced (hexahedral) volume bounded by eight neighboring data points. For finite-element datasets, a cell is equivalent to an element.

The forms of blanking are as follows:

- **Value Blanking** - Cells (or portions of cells) of selected fieldmaps or line plot mappings are excluded based on the value of the value-blanking variable at the data point of each cell or at the point where each cell intersects with a constraint boundary.
- **IJK Blanking** - Cells of one IJK-ordered fieldmap are included or excluded based on the index values. (IJK-ordered fieldmaps only)
- **Depth-Blanking** - Cells in a 3D plot are visually excluded based on their distance from the viewer plane. (3D fieldmaps only)

All types of blanking may be used in a single plot. They are cumulative: cells blanked from any of the options do not appear. Value Blanking and Depth Blanking affect selected fieldmaps of all types, while IJK-blanking affects a single IJK-ordered fieldmap.



Example 51: Depth Blanking

The following sample code activates Depth Blanking at 35%:

```
StyleValue sv(SV_BLANKING, SV_DEPTH);
sv.set( TRUE, SV_INCLUDE);
sv.set( 35.0, SV_FROMFRONT);
```

13 - 9 Frame Linking



Use the [Code Generator](#) to determine the syntax for any of the style settings discussed in this section.

The frame linking feature allows you to link specific style attributes either *between* frames or *within* a frame. Linking between frames allows you to quickly make changes in one frame and propagate them through a number of other frames. Linking within frames links attributes between similar objects within a frame. Refer to [Section 2 - 3 "Frames"](#) for background information on frames.

Linking Style Attributes between Frames

You can use frame linking to apply settings to groups of frames simultaneously. You can specify up to 20 frame linking groups. To setup frame linking between frames, do the following for each frame for which you wish to activate linking:

1. Make the frame active.
2. Specify the group number to use for the frame.
3. Specify any or all of the following attributes between frames:
 - **Frame Size and Position** - Use this option to overlay transparent frames.
 - **Solution Time** - Displays the same time step in all linked frames.
 - **X-axis Range, Y-axis Range** (For XY Line and 2D plots) - Links the X-axis or Y-axis range and the positioning of the left and right sides of the viewport.

- **XY-axis Position** (For XY Line and 2D plots) - Links the positioning of the X- and Y-axes between frames, including the method used for positioning the axes, such as aligning with an opposing axis value.
- **Polar Plot View** - Link views for frames using the Polar Line plot type.
- **3D Plot View** - Link the 3D axes and 3D view.
- **Slice Positions** - Link slice positions and slice planes for active slices (but not slice style).
- **Iso-Surface Locations** (for 3D Plots) - Link iso-surface values (but not iso-surface plot style).
- **Contour Levels** - Link the values and number of contour levels for 2D and 3D plots.
- **Value Blanking Constraints** - Link all value blanking attributes.

Use the [Code Generator](#) to determine the syntax for frame linking with the `StyleValue` class.

After you specify the frame linking settings and group number for each frame, any changes you make in a linked frame will be propagated to the other members of that group. NOTE: Every member of a given group does not have to link the same attributes. For example, if frames 3, 4 and 5 are all members of group 2, but only frames 3 and 5 have the 3D View linked, the 3D View will change in frame 5 only when the view is shifted in frame 3 (and visa versa).



Example 52: Linking 3D Views between Frames

The following example code illustrates linking the 3D View between frames. This example assumes you have two frames established, both showing a 3D plot:

```
TecUtilFrameLightweightPopStart();
TecUtilFrameLightweightPopNext();
StyleValue sv1( SV_LINKING, SV_BETWEENFRAMES, SV_LINK3DVIEW);
sv1.set( TRUE );
TecUtilFrameLightweightPopNext();
TecUtilFrameLightweightPopEnd();
sv1.set( TRUE);
```

Frame Linking Groups

Frames can be segregated into groups so that changes in the linked attributes are propagated only to members of that group. By default, all frames are added to *Group 1*. You can assign a frame to a group

using the `StyleValue` class. New frames added to a group take on the characteristics of previous members of the group.

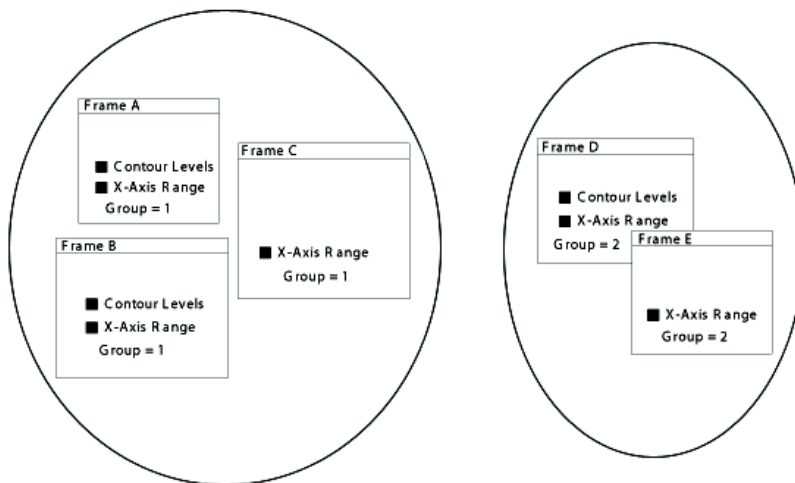


Figure 13-17. Five frames in two groups with different linking options.

Linking Style Attributes Within a Frame

Within the current frame, you can also specify to link the following attributes:

- **Axis Style** - Link activation, colors, line styles, font styles for objects associated with axes.
- **Gridline Style** - Link activation, colors, line styles for gridlines.
- **Zone/Map Color between Plot Layers** - Link the color of meshes, contour lines, and other zone layers for Cartesian plots, or link the color of lines, symbols, and other map layers for line plots.
- **Zone Line Pattern between Plot Layers** - Link line pattern style and length for meshes, vector, and contour lines for Cartesian plots.



You can apply the above settings to multiple frames simultaneously by selecting the group of frames. Refer to [Chapter 21: "Working with Picked Objects"](#) for instructions for identifying the selected frame(s).



Example 53: Linking Axis Style

The following example illustrates linking axis style between two existing frames:

```
TecUtilFrameLightweightPopStart();
TecUtilFrameLightweightPopNext();
StyleValue sv(SV_LINKING, SV_BETWEENFRAMES);
sv.set(TRUE, SV_LINKXAXISRANGE);
```

```
sv.set(TRUE, SV_LINKYAXISRANGE);
sv.set(TRUE, SV_LINKAXISPOSITION);

TecUtilFrameLightweightPopNext();
TecUtilFrameLightweightPopEnd();
sv.set(TRUE, SV_LINKXAXISRANGE);
sv.set(TRUE, SV_LINKYAXISRANGE);
sv.set(TRUE, SV_LINKAXISPOSITION);
```

13 - 10 Animation



While the [Code Generator](#) will output code for animation, it is more efficient to use the TecUtil functions for animation.

You can create Flash, Raster Metafile, and AVI animation files using the following types of animation plots:

- **IJK-plane Animation** - You can display all or a specified subset of the IJK-planes in your dataset, one at a time using `TecUtilAnimateIJKPlanesX`. You can choose to animate either the I, J, or K-planes.
- **IJK-blanking Animation** - Animate a sequence of renderings starting with an initial set of blanked IJK indices and proceeding in a series of interpolated steps to a final set of blanked IJK indices using `TecUtilAnimateIJKBlankingX`. Before you can animate IJK-blanking, you must first activate IJK-blanking. Refer to [Section 13 - 8 "Blanking"](#) for details.
- **Iso-surfaces Animation** - Animate iso-surfaces either on screen or to a file. You must specify a starting value, an ending value, and the number of steps using `TecUtilAnimateIsoSurfaces`. If you specify a start level with a higher number than the end level, the Tecplot Engine cycles backward from the start to the end.
- **Mapping Animation** - Display all or a specified subset of the XY or Polar Line mappings defined in the current frame, one at a time using `TecUtilAnimateLineMapsX`.
- **Slice Animation** - Animate slices either on screen or to a file using `TecUtilAnimateSlicesX`. You must specify a starting value, an ending value, and the number of steps. If you specify a start level with a higher number than the end level, the Tecplot Engine cycles backward from the start to the end.
- **Streamtrace Animation** - Animate slices either on screen or to a file using `TecUtilAnimateStreamX`. You must specify the number of steps per cycle and the number of cycles.
- **Zone Animation** - Animate zones/fieldmaps either on screen or to a file using `TecUtilAnimateZonesX`. This option is not available for transient datasets.
- **Time Animation** - Animate transient data using `TecUtilAnimateTimeX`. See also [Section 13 - 11 "Working with Transient Data"](#).

Use the [Code Generator](#) to determine the syntax for any of the above animation styles.



Example 54: Animating Iso-surfaces

The following sample code illustrates how to animate iso-surfaces in an existing 3D Plot containing volume data:

```
Arglist arglist;  
arglist.appendDouble( SV_STARTVALUE      , -0.95);  
arglist.appendDouble( SV_ENDVALUE        , 1.95);  
arglist.appendInt(    SV_NUMSTEPS        , 25);  
arglist.appendInt(    SV_LIMITSCREENSPEED, TRUE);  
arglist.appendDouble( SV_MAXSCREENSPEED   , 10.6);  
TecUtilAnimateIsoSurfacesX( arglist.getRef() );
```

13 - 11 Working with Transient Data

For transient datasets, you can display your data at a given time or to animate your data over time. The loaded zones can be linked to a specific solution time, and the active solution time is used to determine which zones are displayed. Refer to [Section 23 - 9 “Transient Data”](#) for information on specifying transient zones when loading your data.

For the following definitions, consider the following fictitious dataset:

Table 13 - 3: Sample Transient Dataset

Zone	Time	StrandID
1	n/a	n/a
2	0.0	2
3	0.18	3
4	0.22	1
5	0.25	2
6	0.28	1
7	0.32	3
8	0.38	2
9	0.42	1
10	0.52	1
11	0.57	2
12	0.58	3
13	0.62	1
14	n/a	n/a

- **Transient zones** - Zones associated with time. The transient zone(s) displayed in the current frame are dependent upon the current solution time. Zones 2-13 in [Table 13 - 3](#) are transient zones.
- **Static zones** - Zones not associated with time. They are displayed regardless of the current solution time. Zones 1 and 14 from [Table 13 - 3](#) are static.
- **Current Solution Time** - The value that determines which transient zones are displayed in the current frame.
- **Strand** - A series of transient zones that represent the same part of a dataset at different times. Zones 2, 5, 8, and 11 in [Table 13 - 3](#) all have the same StrandID and, therefore, are part of the same strand.
- **StrandID** - An integer value defined for each transient zone. The StrandID of a given zone is determined by the data loader.

- **Relevant Zone** - Only “relevant zones” are plotted at a given solution time. A relevant zone is defined as a zone for a given strand used for a certain solution time. If the strand exists at solution time n , the relevant zone is either the transient zone on that strand defined explicitly at solution time n , or the zone defined immediately prior to solution time n . If the strand does not exist at solution time n , there are no relevant zones for that strand at that time. Static zones are always considered relevant zones. [Figure 13-18](#) below illustrates how relevant zones are determined.

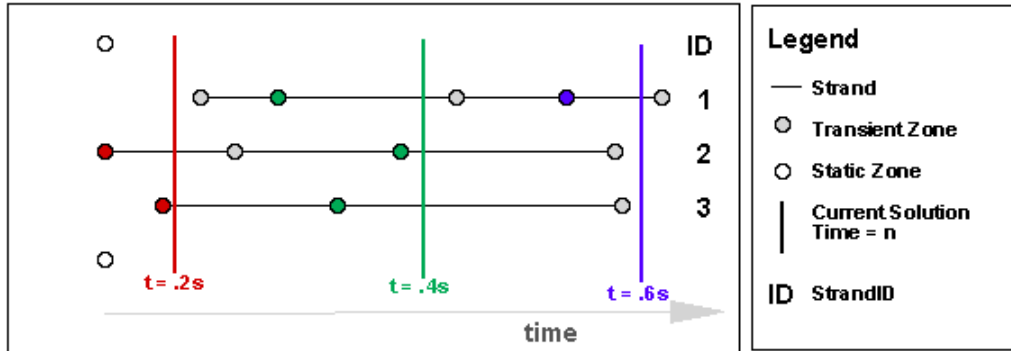


Figure 13-18. An illustration of how relevant zones are determined (based on the data in [Table 13 - 3](#)). For a given solution time, ONLY the relevant zones are displayed in the plot. NOTE: static zones are always considered relevant zones.

$t = .2s$ - The red-colored transient zones and both static zones are plotted. NOTE: no zones from the first strand are represented because the strand is not defined at that time.

$t = .4s$ - The green-colored transient zones and both static zones are plotted.

$t = .6s$ - The blue-colored transient zones and both static zones are plotted. NOTE:



Example 55: Animating Transient Data

The following example code illustrates creating a time animation:

```
TecUtilOpenLayout( DataFile.c_str(), NULL, FALSE);

ArgList_pa ArgList;
ArgList = TecUtilArgListAlloc();
TecUtilArgListAppendInt(ArgList,SV_CREATEMOVIEFILE,TRUE);
TecUtilArgListAppendInt(ArgList,SV_EXPORTFORMAT,ExportFormat_AVI);
TecUtilArgListAppendString(ArgList, SV_FNAME, "Time.avi");
TecUtilArgListAppendDouble(ArgList, SV_STARTTIME, 0.000000);
TecUtilArgListAppendDouble(ArgList, SV_ENDTIME, 0.002419);

TecUtilAnimateTimeX(ArgList);
TecUtilArgListDealloc(&ArgList);
```

13 - 12 View Options



The View options provided by the Tecplot Engine are not accessible via the [Code Generator](#) and the StyleValue class. You will need to use the along with the following section to enable view control.

You can adjust the view in the current frame using the following options:

- [Rotation](#)
- [Translation](#)
- [Zoom](#)

13 - 12.1 Rotation

You may rotate your field plots using either [2D Rotation](#) or [3D Rotation](#).

2D Rotation

Two-dimensional field data can be rotated about a user specified XY-origin. Two-dimensional data rotation can be performed with the TecUtilDataRotate2D function.

The prototype for TecUtilDataRotate2D is:

```
Boolean_t TecUtilDataRotate2D ( Set_pa ZoneSet,
                                double RotateAmountInDegrees,
                                double XOrigin,
                                double YOrigin
                                )
```

Where the parameters are described as:

ZoneSet	Specify the zones to rotate.
RotateAmountInDegrees	Specify the angle of rotation, in degrees.
XOrigin	Specify the coordinates of the X-origin of rotation.
YOrigin	Specify the coordinates of the Y-origin of rotation.

3D Rotation

Use TecUtilViewRotate3D to work with three-dimensional rotation. You can perform three-dimensional rotation about any of the following rotation axes:

- **X, Y, or Z-Axis** - Rotation about one of the three axes: X, Y, or Z.
- **Psi, Theta or Alpha-Axis** - Spherical rotation about the Psi, Theta or Alpha axis.
- **Horizontal or Vertical RollerBall** - Rotation like a roller ball, that is horizontal movements, are right and left from the current position; vertical movements rotate up or down from the current position; and twist is about the current screen Eye/Origin ray.
- **Vector** - Specify a vector to rotate about.

- **Center of Rotation** - Use this option to set the center of rotation to be the Center of Data (the center of the bounding box of the data) or Center of View (the point hit by a probe at frame coordinates 50%, 50%).



Center of View can result in an error if there is no data in the center of the frame. If this is the case, the center of rotation will not move.



Example 56: Rotating the Plot

The following sample code illustrates how to rotate the plot by 65% about the Z-axis translated to the currently defined origin:

```
TecUtilViewRotate3D( RotateAxis_Z,
                     0.65*360,
                     0, 0, 0,
                     RotateOriginLocation_DefinedOrigin);
```



Example 57: Changing the Center of Rotation

The following sample code illustrates how to change the center of rotation to 1,1,1:

```
StyleValue sv( SV_GLOBALTHREED, SV_ROTATEORIGIN);
sv.set( 1.0, SV_X);
sv.set( 1.0, SV_Y);
sv.set( 1.0, SV_Z);
```

13 - 12.2 Translation

You can translate your display in the following ways:

- **Plot** - Use TecUtilViewTranslate to shift the plot in the X and/or Y direction.
- **WorkView** - Use TecUtilWorkViewTranslate to shift the view of the workspace.

13 - 12.3 Zoom

Similar to translation, you can zoom into either the plot or the workspace using TecUtilViewZoom or TecUtilWorkViewZoom, respectively. Use TecUtilViewSetMagnification to set the magnification factor.



Example 58: Adjusting the Magnification Factor

The following sample code illustrates how to change the Magnification Factor to 0.85:

```
TecUtilViewSetMagnification(0.85);
```

Code Generator

In order to set or get a Tecplot 360 style value, you must know the hierarchy of the value in which you are interested, as well as the available style values. The hierarchy of the value can be determined using the Code Generator add-on. The values available for a given parameter can be determined using the Code Generator, coupled with the [ADK Reference Manual](#).

The variety of style option available are discussed in detail in [Chapter 13: "Setting Plot Style"](#).

14 - 1 Style Hierarchy

The Tecplot Engine uses an extensive style hierarchy to isolate each plot attribute, with attributes such as fieldmap, linemap, interface, global paper and sketch axis at the top level of the hierarchy. Items such as plot type, plot layer, and derived object (i.e. iso-surface, slice or streamtrace) are immediately below the top level, followed by a characteristic of the secondary item.

For example, the contour style or type is a secondary characteristic of the contour plot layer (which is an attribute of a fieldmap). As such, the data hierarchy for ContourType (e.g. lines or flood) is:

FIELDMAP, CONTOUR, CONTOURTYPE

The following list shows the hierarchy for the style values for the main "Axis Line" for the Y-Axis in 2D plots¹:

```
$!TwoDAxis YDetail {AxisLine {Show = (Boolean_t)}}
$!TwoDAxis YDetail {AxisLine {ShowPerpendicular = (Boolean_t)}}
$!TwoDAxis YDetail {AxisLine {ShowBothDirections = (Boolean_t)}}
$!TwoDAxis YDetail {AxisLine {ShowOppositeEdge = (Boolean_t)}}
$!TwoDAxis YDetail {AxisLine {Color = (ColorIndex_t)}}
$!TwoDAxis YDetail {AxisLine {LineThickness = (double)}}
$!TwoDAxis YDetail {AxisLine {Alignment = [AxisAlignment_e]}}
$!TwoDAxis YDetail {AxisLine {OpposingAxisValue = (double)}}
$!TwoDAxis YDetail {AxisLine {Position = (double)}}
$!TwoDAxis YDetail {AxisLine {Angle = (double)}}
$!TwoDAxis YDetail {AxisLine {Offset = (double)}}
$!TwoDAxis YDetail {AxisLine {Edge = (SmInteger_t)}}
```

1. An exhaustive list of the style hierarchy can be viewed by examining the comlist file supplied with your distribution. This file lists the hierarchy in the form of the corresponding native macro command syntax including the assignment type for each entry. The comlist file is located in the home directory.

In general, a style command contains the following information:

Style Navigation List	List of commands to traverse the style hierarchy
primary offset	(if applicable) Offset needed to access the n th item.
secondary offset	(if applicable) Secondary Offset.
ItemSet	(if applicable) Set of items to apply the style to (e.g. set of linemaps)
new value	New value to apply (e.g. color, line thickness value, etc.)

The Code Generator add-on is provided to help you determine the style hierarchy for a plot attribute you wish to expose. The overview of the Tecplot Engine style hierarchy in this section is provided for your convenience.



Example 59: Changing the Line Thickness

The StyleValue command needed to change the line thickness of the Y-Axis in a 2D plot to 1.5 is:

```
StyleValue styleValue;  
styleValue.set(1.5, SV_TWODAXIS, SV_XDETAIL,  
               SV_AXISLINE, SV_LINETHICKNESS);
```

The "SV" constants mimic the macro command navigation options with a "SV_" prefix.



Example 60: Using an Offset with the StyleValue Class

In some cases a set or offset (or two offsets) are also needed to identify the exact entity to modify. For example, in line plots there can be multiple X- or Y-Axes. Use the following code to set the line thickness for the third Y-Axis in a line plot:

```
styleValue.set(2,3,SV_XYLINEAXIS,SV_XDETAIL,SV_AXISLINE,  
               SV_LINETHICKNESS);
```



Example 61: Using Sets with the StyleValue Class

Most of the time you will be applying style to just one entity. In some cases the style can be applied to a set of entities with one command. This is true for fieldmaps and linemaps. For example, use the following code to change the mesh color for the first three fieldmaps to Green.

```
Set objectSet("[1-3]");
styleValue.set(Green_C, objectSet, SV_FIELDMAP, SV_MESH, SV_COLOR);
```

14 - 2 Code Generator

The Code Generator is an add-on that displays the style value code corresponding to any plot style changes you have made. You have the option to display the code using either the Tecplot Toolbox StyleValue class, Python, or classic code (which uses a series of TecUtil function calls). We highly recommend that you use the Tecplot Toolbox StyleValue class, as it is much easier to use than the classic code. Not only does the StyleValue class handle memory management issues, it also requires a fraction of the code. If you are an add-on developer working with the Tecplot 360, you may be more comfortable with the classic TecUtil code. However, you will probably find the StyleValue class to be much easier to use.

14 - 2.1 Launching the Code Generator

The Code Generator add-on is part of the installation included with your Tecplot 360 distribution. To use the Code Generator, simply select "Code Generator..." from the **Tools** menu.

Once the Code Generator dialog is displayed, simply create¹ a plot and make a style change that duplicates a behavior you would like to have. The changes you make via the interface will appear as C++ code in the Code Generator dialog. Copy and paste the code from the dialog to your source. Modify the source code using the instructions described in [Section 14 - 2.2 "Incorporating Code Generator Output"](#) below.



The Code Generator will not produce code for interactive style changes, such as using the mouse to rotate or translate, or manipulating data, etc. To perform these actions, you must search through the [ADK Reference Manual](#) for the TecUtil function(s) that meet your needs.

14 - 2.2 Incorporating Code Generator Output

Once you have changed the value of the attribute you wish to give your users control over, copy-and-paste the Code Generator output, and enhance the output as needed.

The generated code is not always complete and in most cases you must make some minor adjustments. The following portions must be modified:

- **Data Values** - If you made a change to a given parameter and wish to provide your users access to any of the possible values of that parameter, you will need to determine the rest of the acceptable values. For example, if you change the contour type for the first fieldmap to flood, the Code Generator output is:

```
objectSet.assign("[1]");
styleValue.set((ContourType_e)2, objectSet, SV_FIELDMAP, SV_CONTOUR,
SV_CONTOURTYPE);
```

1. When working with the Code Generator, it is imperative that Tecplot 360 be in the state your end users will be in before you generate the code. Otherwise, the results may not be as expected. For details on creating and customizing plots in Tecplot 360, please refer to the [Tecplot 360 User's Manual](#) (included in with your Tecplot 360 distribution).

To set the “ContourType” parameter to ContourType_Flood, the code is:

```
styleValue.set(ContourType_Flood, objectSet, SV_FIELDMAP, SV_CONTOUR,  
SV_CONTOURTYPE);
```

Refer to the [ADK Reference Manual](#) for the list of available values for any data type.

- **Sets** - If you made a change to a single zone, but you wish to allow your users to make the change to an arbitrary set of zones, you will need to adjust the set according.

For example, the following code sets the line color for the third fieldmap.

```
Set objectSet("[3]");  
styleValue.set((ColorIndex_t)3, objectSet, SV_FIELDMAP, SV_MESH,  
SV_COLOR);
```

You will need to make accommodations for the object set and color to be determined via user input. See also [Chapter 24: “Sets”](#).

- **Offsets** - Some style items are part of an array (or two-dimensioned array) of settings. For example, iso-surface settings are assigned to a particular iso-surface group. In order to modify iso-surface style, you must provide the group number as the offset.

For example, with the Code Generator running, activate iso-surfaces for group 3 and then elect to display the mesh on the iso-surfaces. The Code Generator will output:

```
StyleValue styleValue;  
styleValue.set((Boolean_t)1, 3, SV_ISOSURFACEATTRIBUTES,  
SV_SHOWGROUP);  
styleValue.set((Boolean_t)1, 3, SV_ISOSURFACEATTRIBUTES, SV_MESH,  
SV_SHOW);
```

To use this in your code change "(Boolean_t)1" to "TRUE":

```
StyleValue styleValue;  
styleValue.set(TRUE, 3, SV_ISOSURFACEATTRIBUTES, SV_SHOWGROUP);  
styleValue.set(TRUE, 3, SV_ISOSURFACEATTRIBUTES, SV_MESH, SV_SHOW);
```

Style Change Prerequisites

When incorporating the Code Generator output into your source code, it is best to monitor the output for the entire sequence of steps leading up to modifying a given attribute. In many cases, there are style settings that are dependent upon the settings made beforehand.

For example, before the vector layer is activated, the vector variables are defined. Make sure the prerequisite steps take place, i.e. setting the current frame or selecting a linemapping. As a rule, variable assignments are required if the Code Generator produces code that assigns variables prior to other settings.

14 - 2.3 Commands Not Displayed by the Code Generator

The following types of set value commands are not displayed by the Code Generator:

- **Text and Geometries** - Text and Geometries are not handled by the Code Generator. To use Text and Geometries, see the TecUtilText and TecUtilGeom families of functions in the [ADK Reference Manual](#).
- **View Settings** - View settings are primarily controlled by the TecUtilView and TecUtilWorkView families of functions. The Code Generator does not generate code for the majority of the view settings. Please consult the [ADK Reference Manual](#) for information on the TecUtilView and TecUtilWorkView families of functions.

The Code Generator will not produce code for interactive style changes, such as using the mouse to rotate or translate, or manipulating data, etc. To perform these actions, you must search through the [ADK Reference Manual](#) for the TecUtil function(s) that meet your needs.

Porting Add-ons

Ideally, the process of transferring an add-on between operating systems begins when you write the first version of your add-on. The cross-platform strategy revolves around creating the original add-on with cross-platform-compatible ingredients. For many users, this means writing the add-on in C, since many Linux machines have C compilers readily available.

If your add-on has a graphical user interface, we recommend you use the [Tecplot GUI Builder](#) (TGB), or another code library which is portable between platforms. If your add-on uses MFC, then you must isolate the MFC code as much as possible and include separate code for Motif®.

15 - 1 Porting Add-ons from a Windows Platform to a Linux Platform

The following procedure outlines the general steps for porting add-ons from a Windows platform to a Linux platform:

1. If your code is in C++ files, create a C file to compile on Linux by doing the following:
 - a. For each *.cpp* file, create a *.c* file with the same name.
 - b. Move all of the code from the *.cpp* file into the *.c* file.
 - c. Delete the code from the *.cpp* file and add the line: `#include "xxx.c"`.
 - d. If this was an MFC project, add the line `#include "stdafx.h"` to the top of the file.
2. In your Linux machine, run the CreateNewAddOn script to start a new add-on. Be sure to select the TGB option if your add-on used the TGB option on a Windows platform. See [Section 2 - 2 "Creating a New Add-on"](#) for more information on how to use CreateNewAddOn.
3. Move all of the *.c* and *.h* files from your Windows project into the Linux project directory.
4. Edit the Linux Makefile to include the new *.c* and *.h* files.
5. Compile your add-on for Linux.

15 - 2 Porting Add-ons from a Linux Platform to a Windows Platform

The general procedure for porting add-ons from a Linux platform to a Windows platform is as follows:

1. Follow the directions in [Chapter 3: “Creating Add-ons on Windows Platforms”](#) for creating a non-MFC DLL in Visual Studio. It is important that you select "Win32 Project" as the project template, and "DLL" as the Application Type in the Application Wizard.
2. Move all of the *.c, *.cpp, and *.h files and *gui.lay* from your Linux project directory to the new project directory on your Windows machine.
3. In Visual Studio, select "Add Existing Item" from the **Project** menu, and add all of the *.c and *.h files. Right-click on the *guibld.c* file in the Solution Explorer window, and open the **Properties** dialog. On the General page of the Configuration Properties, set the toggle "Excluded file from Build" to "Yes". Select [OK].
4. Link with *wingui.lib*.
5. Select **Build>Build Solution** to build your add-on.

Part 4 Lists and Sets

Argument Lists

Several `TecUtil` functions require a flexible, or extended, argument list. The `ArgList` class in the Tecplot Toolbox is provided to handle these `TecUtil` functions. `ArgLists` provide a method to supply a variable set of name-value pairs when calling a function. Most of these values are optional.

When an extended function is created or as new capabilities are added to an existing extended function, reasonable default values are assigned whenever appropriate. The defaults also provide backward compatibility for written and compiled with prior versions of Tecplot.

All `TecUtil` functions that use extended argument lists end with the capital letter `X`, distinguishing them from standard argument list functions. Where appropriate, a standard argument list function is provided along with the extended version so that common uses of the function are not burdened with the additional instructions required for the extended version.

16 - 1 Tecplot Toolbox ArgList Class

The Tecplot Toolbox provides the `tecplot::toolbox::ArgList` class to manage the `ArgList_pa` type. To use the Tecplot Toolbox, you must include `tptoolbox.h` in your project.



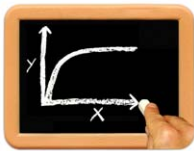
Example 62: Using the ArgList Class with TecUtilDataSetAddZoneX

```
ArgList argList;  
argList.appendString(SV_NAME, "New Zone");  
argList.appendInt(SV_ZONETYPE, ZoneType_Ordered);  
argList.appendInt(SV_IMAX, 10);  
TecUtilDataSetAddZoneX(argList.getRef());
```

16 - 2 TecUtil Functions that Use Argument Lists

Currently, argument lists are used in the following TecUtil functions:

```
TecUtilSaveLayoutX
TecUtilReset3DOriginX
TecUtilAnimateZonesX
TecUtilAnimateContourLevelsX
TecUtilAnimateIJKPlanesX
TecUtilAnimateIJKBlankingX
TecUtilAnimateStreamX
TecUtilAnimateSlicesX
TecUtilImageBitmapCreateX
TecUtilDataSetAddZoneX
TecUtilAnimateLineMapsX
TecUtilStateChangedX
TecUtilTransformCoordinatesX
TecUtilZoneCopyX
TecUtilCreateSliceZoneFromPlaneX
```



Example 63: Using Argument Lists

Following is an example of how to set up and use the argument lists:

```
ArgList_pa      ArgList = NULL;
Boolean_t       LayoutSaved = FALSE;
Boolean_t       IncludeData = FALSE;
Boolean_t       IncludePreview = FALSE;
Boolean_t       UseRelativePaths = FALSE;
char            LayoutFName[MAX_FNAME_LEN+1];
/* Get the layout save options from the user. */
.
.
.
/* Create the argument list and initialize it with the save options. */
ArgList = TecUtilArgListAlloc;
if (ArgList != NULL)
{
    TecUtilArgListAppendString(ArgList,SV_FNAME,LayoutFName);
    TecUtilArgListAppendInt(ArgList,SV_INCLUDEDDATA,IncludeData);
    if (IncludeData)
        TecUtilArgListAppendInt(ArgList,SV_INCLUDEPREVIEW,
IncludePreview);
    else
        TecUtilArgListAppendInt(ArgList, SV_USERRELATIVEPATHS,
UseRelativePaths);

/* Save the layout and cleanup the argument list. */
    LayoutSaved = TecUtilSaveLayoutX(ArgList);
    TecUtilArgListDealloc(&ArgList);
.
.
.
```

String Lists

A *string list* is simply a list or collection of strings. The `StringList_pa` type is simply a handle to a string list maintained by the Tecplot Engine. There are several `TecUtil` functions provided to handle the `StringList_pa` type, and a number of `TecUtil` functions require the `StringList_pa` type as a parameter. For example, string lists are used when loading data - the list of data files to load is contained in a string list.

The Tecplot Toolbox provides a class, `tecplot::toolbox::StringList`, which deals with the `StringList_pa` type. The `StringList` class manages the `StringList_pa` type in such a way that the client generally does not have to worry about memory management issues as you would when using the legacy `TecUtilStringList` family of functions. To use the Tecplot Toolbox, you must include `tptoolbox.h` in your project.

A notable exception is when the `StringList_pa` type is an output parameter or return value of a `TecUtil` function, then the `StringList_pa` must be deallocated using `TecUtilStringListDealloc`.

17 - 1 Tecplot Toolbox StringList Examples

The following are simple examples of creating `StringList` objects. For more detailed information on the `StringList` class, see the [ADK Reference Manual](#).



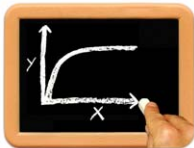
Example 64: Use a StringList to Create a Dataset

```
// It is unnecessary to use both constructor and append method to
// populate the StringList. However, it illustrates a couple
// different ways of populating a StringList.
tecplot::toolbox::StringList varNames("X", "Y", NULL);
varNames.append("Z");
TecUtilDataSetCreate("My DataSet",
                    varNames.getRef(),
                    TRUE); // ResetStyle
```



Example 65: Working with StringList_pa as an Output Parameter

```
char *loaderName;
StringList_pa tmpInstructions;
TecUtilImportGetLoaderInstr(&loaderName,
                           &tmpInstructions);
// Assign tmpInstructions to loaderInstructions
tecplot::toolbox::StringList loaderInstructions(tmpInstructions);
TecUtilStringDealloc(&loaderName);
TecUtilStringListDealloc(&tmpInstructions);
... do something with loaderInstructions ...
```



Example 66: Using String Lists

Following is an example of how to set up and use string lists:

```
int          i = 0;
int          count = 0;
char         *file_name = NULL;
Boolean_t    selected = FALSE;
StringList_pa selected_file_names = NULL;
StringList_pa default_file_names = NULL;
.
.
.
/* Set up the default file name list. */
default_file_names = TecUtilStringListAlloc;
TecUtilStringListAppendString(default_file_names, "a.dat");
TecUtilStringListAppendString(default_file_names, "b.dat");

/* Ask user to select a bunch of add-on data files. */
selected = TecUtilDialogGetFileNames(SelectFileOption_ReadMultiFile,
                                     &selected_file_names, "Addon Data",
                                     default_file_names, "*.dat");
/* We do not need the default list of file names any more. */
TecUtilStringListDealloc(&default_file_names);
/* Process the results (for simplicity print to standard output). */
if (selected)
{
    /* Ask the string list how many string items it maintains. */
    count = TecUtilStringListGetCount(selected_file_names);
    /* Print the header information. */
    printf("You selected the following files:\n");
    printf("-----\n");
```

```
/* Print each one to standard output. */
for (i = 1; i <= count; count++)
{
    file_name =
        TecUtilStringListGetString(selected_file_names,i);
    printf(" %s\n", file_name);
    /* Deallocate return value as it is no longer needed. */
    TecUtilStringDealloc(&file_name);
}
/* We do not need the list of selected file names any more. */
TecUtilStringListDealloc(&selected_file_names);
}
.
.
.
```


Sets

A *set* is a group or collection of zones, variables, or line-mappings. The set is a convenient way of dealing with groups of numbers. Each number that exists in a set is referred to as a member of the set. The `Set` class in the Tecplot Toolbox is provided to deal with this type.

A set will not accept a value that already exists within it. In other words, if the set already contains a value of 2, adding another 2 does not change the list of values or the count of total values. This ensures a compact set without duplicates. Sets are also sorted and are limited to values of greater than or equal to one.

In many cases where a set is used, you may pass `NULL` to indicate all values. For example, `TecUtilReadDataSet` takes a parameter which is the set of zones you want to read. If you pass `NULL` for this parameter, all zones are loaded. To determine whether you can use `NULL` in this manner, refer to the . When using the `tecplot::toolbox::Set` class provided by the Tecplot Toolbox, you must explicitly pass `NULL` rather than using the `Set::getRef` method, since the `Set` class always maintains a non-`NULL` `Set_pa` handle.



Refer to sample add-on *modifydata* for an example of working with sets.

18 - 1 Tecplot Toolbox Set Class

The `tecplot::toolbox::Set` class provided by the Tecplot Toolbox manages the `Set_pa` type such that the client typically does not have to worry about allocation or deallocation. A notable exception is when the `Set_pa` type is an output parameter or return value of a `TecUtil` function, the `Set_pa` may have to be deallocated using `TecUtilSetDealloc`. Refer to the for specifics on whether the `TecUtil` function returns a set that must be deallocated or not. To use the Tecplot Toolbox, you must include *tptoolbox.h* in your project.

Note that the `Set` constructor which takes a `Set_pa` type copies the contents of the `Set_pa` rather than assuming ownership of it.



Example 67: Getting the Set of Enabled Zones

```
Set_pa zoneSet;
TecUtilZoneGetEnabled(&zoneSet);
Set enabledZones(zoneSet);
TecUtilSetDealloc(&zoneSet);
if ( enabledZones.isMember(3) )
{
    // Do something to zone 3
}
```



Example 68: Looping Through the Members of a Set

```
// Assume we have a set populated with zone numbers
LgIndex_t zone;
TecUtilSetForEachMember(zone, zoneSet.getRef())
{
    if ( TecUtilZoneIsFiniteElement(zone) )
    {
        ... do something with zone ...
    }
}
```



Example 69: An Alternate Way of Looping through Members of a Set

```
LgIndex_t zone = zoneSet.getNext(TECUTILSETNOTMEMBER);
while ( zone != TECUTILSETNOTMEMBER )
{
    if ( TecUtilZoneIsFiniteElement(zone) )
    {
        ... do something with zone ...
    }
}
```

```

    zone = zoneSet.getNext(zone);
}

```

18 - 2 TecUtil Functions that Use Sets

Sets are used in the following TecUtil functions:

```

TecUtilReadDataSet
TecUtilWriteDataSet
TecUtilDataAlter
TecUtilCreateMirrorZones
TecUtilPolarToRectangular
TecUtilRotate2D
TecUtilDataRotate2D
TecUtilAverageCellCenterData
TecUtilInverseDistInterpolation
TecUtilKrig
TecUtilLinearInterpolate
TecUtilLineMapGetActive
TecUtilLineMapSetActive
TecUtilLineMapSetName
TecUtilLineMapSetAssignment
TecUtilLineMapSetLine
TecUtilLineMapSetCurve
TecUtilLineMapSetSymbol
TecUtilLineMapSetSymbolShape
TecUtilLineMapSetBarChart
TecUtilLineMapSetErrorBar
TecUtilLineMapSetIndices
TecUtilLineMapDelete
TecUtilLineMapShiftToTop
TecUtilLineMapShiftToBottom
TecUtilTriangulate
TecUtilProbeAtPosition
TecUtilVarGetEnabled
TecUtilPickAddZones
TecUtilPickAddMaps
TecUtilZoneDelete
TecUtilZoneGetActive
TecUtilZoneGetEnabled
TecUtilZoneSetActive
TecUtilZoneSetMesh
TecUtilZoneSetContour
TecUtilZoneSetVector
TecUtilZoneSetVectorIJKSkip
TecUtilZoneSetScatter
TecUtilZoneSetScatterIJKSkip
TecUtilZoneSetScatterSymbolShape
TecUtilZoneSetShade
TecUtilZoneSetBoundary
TecUtilZoneSetVolumeMode

```



Example 70: Using Sets

The following example shows uses of all of the TecUtilSetxxx functions:

```
/*
 * Create two sets, A and B. A will have 1,2,3 for its
 * members, and B will have 4 and 9.
 */
Set_pa A;
Set_pa B;
A = TecUtilSetAlloc(TRUE);
B = TecUtilSetAlloc(TRUE);
if (A && B)
{
    SetIndex_t Position, Member;
    /*
     * Add the members to the sets.
     */
    TecUtilSetAddMember(A,1,TRUE);
    TecUtilSetAddMember(A,2,TRUE);
    TecUtilSetAddMember(A,3,TRUE);
    TecUtilSetAddMember(B,4,TRUE);
    TecUtilSetAddMember(B,9,TRUE);

    /*
     * Check to see if the sets are equal.
     */
    if (TecUtilSetIsEqual(A,B))
        TecUtilDialogErrMsg("Something is wrong here");

    /*
     * Clear out set A.
     */
    TecUtilSetClear(A);
    /*
     * Make A a copy of B.
     */
    TecUtilSetCopy(A,B,TRUE);

    /*
     * Get the position of the member '9' of set B
     * (the result is '2').
     */
    Position = TecUtilSetGetPosition(B,9);

    /*
     * Get the member located at position '2' of set A
     * (the result is '4').
     */
    Member = TecUtilSetGetMember(A,Position);

    /*
     * Get the member located after the member '4' of
     * set A (the result is '9').
     */
    Member = TecUtilSetGetNextMember(A,Member);

    /*
     * Remove the first valid member from B.
     */
    if (TecUtilSetGetMemberCount(B) > 0)
    {
        int I = 1;
        while (!TecUtilSetIsMember(B,I))
            I++;
        TecUtilSetRemoveMember(B,I);
    }
}
```

```
    }  
  
    /*  
    * Show a warning dialog if B is now empty.  
    */  
    if (TecUtilSetIsEmpty(B))  
        TecUtilDialogMessageBox("B is empty", MessageBox_Warning);  
  
    /*  
    * Finally, deallocate the sets.  
    */  
    TecUtilSetDealloc(&A);  
    TecUtilSetDealloc(&B);  
}
```


Part 5 Additional Functionality

Augmenting the Macro Language of Tecplot 360

Tecplot 360's macro language allows you to automate tasks that are performed repeatedly. Macro functions can be assigned to buttons in the Quick Macro Panel, or used to load and process data retrieved from a large number of files. You can augment Tecplot 360's macro language with your add-on's own macro functions, allowing your add-on's tasks to be automated as well.

19 - 1 Processing Custom Macro Commands

You can augment Tecplot 360's macro language with your own set of commands that will be routed directly to your add-on for processing. The Tecplot 360 macro command `$!EXTENDEDCOMMAND` bridges Tecplot 360's macro language and your add-on. The `$!EXTENDEDCOMMAND` macro command syntax is:

```
$!EXTENDEDCOMMAND
COMMANDPROCESSORID = string
COMMAND = string
```

where the string assigned to `COMMANDPROCESSORID` identifies which add-on is to receive the command, and the `COMMAND` parameter identifies the commands to be processed by the add-on.

To inform Tecplot 360 of the function to call when it encounters your macro command, call `TecUtilMacroCommandExtCallback` from the `InitTecAddOn` function in your add-on. The call to `TecUtilMacroCommandExtCallback` is defined as:

```
TecUtilMacroCommandExtCallback(MyCommandProcessorID, MyMacroProcessor);
```

Where `MyCOMMANDPROCESSORID` is a string used in the `COMMANDPROCESSORID` part of the `$!EXTENDEDCOMMAND` and `MyMacroProcessor` is the name of a function you write to handle the macro command.



Example 71: Zone Processing

This example uses an add-on that sums the areas (or volumes) of all cells in a specified list of zones. You want to create a macro command that can tell your add-on which zones to process.

The first task is to create a function in your add-on that can handle these instructions. This function will look something like:

```
Boolean_t STDCALL ProcessSumCellsCommand(char *CommandString,
                                         char **ErrMsg)
{
    Boolean_t IsOk = TRUE;
    /*
     * Process commands in CommandString
     */
    return (IsOk);
}
```

The next task is to add the following line to the InitTecAddOn function in your add-on:

```
...
TecUtilMacroCommandExtCallback("SUMCELLS",
                               ProcessSumCellsCommand);
...
```

This tells Tecplot 360 to watch out for macro commands that look like

```
$!EXTENDEDCOMMAND
COMMANDPROCESSORID = "SUMCELLS"
COMMAND = "1,2,5-9"
```

When Tecplot 360 processes a command similar to the one shown, it calls the function you registered with the second parameter to TecUtilMacroCommandExtCallback, which in this case is ProcessSumCellsCommand. ProcessSumCellsCommand is called with the command which is the string taken from the COMMAND parameter in the macro. In the preceding example, ProcessSumCellsCommand would be called with the string "1,2,5-9" as its first parameter.

You may design any syntax you wish for the instructions sent to your add-on. The only restriction is that they must be able to fit into a single string in a Tecplot 360 macro sub-command. In the previous example, the function ProcessSumCellsCommand will be coded so that it can scan a comma and dash-delimited set of numbers and determine a set of zones for which to sum the areas or volumes.

19 - 2 Error Processing

If your macro command callback function detects an error during processing or in the command string, it is required to do the following:

1. Allocate enough space for an error message by calling TecUtilStringAlloc. The ErrMsg parameter to the callback function must be assigned to this space.
2. Generate an appropriate error message and place it into the space created in Step 1.
3. Return with a value of FALSE.

For example, if the function ProcessSumCellsCommand in the previous section detects that a zone specified in the command does not exist (and this is determined to be an error condition). The coding for ProcessSumCellsCommand may then look like:

```
Boolean_t STDCALL ProcessSumCellsCommand(char *CommandString,
                                         char **ErrMsg)
{
    Boolean_t      IsOk = TRUE;
    EntIndex_t    CurZone;
    char          *CPtr = CommandString;
```

```

double                               SumTotal = 0;

/*
 * Scan CommandString and pull out zones to sum. The function
 * GetNextZone pulls out the next zone number and advances CPtr.
 * SumNextZone attempts to calculate a sum in the next zone and
 * returns FALSE if the zone requested is invalid.
 */

while (IsOk && GetNextZone(&CPtr,&CurZone))
{
    double CurSum;
    if (SumNextZone(CurZone,&CurSum))
        SumTotal += CurSum;
    else
    {
        IsOk = FALSE;
        *ErrMsg = TecUtilStringAlloc(200);
        sprintf(*ErrMsg,"Can't calc sum for zone %d",CurZone);
    }
}
return (IsOk);
}

```

The functions `GetNextZone` and `SumNextZone` are not provided for brevity.

19 - 3 Recording Custom Macro Commands

If the end-user is recording a macro that uses your add-on, you will want the action to be translated into a macro command written to the macro record file.

To record a macro command, call the function `TecUtilMacroRecordExtCommand` after your add-on has successfully performed an operation requested by the user.

For example, the user, via dialogs in your add-on, requests to sum the areas of cells in zones 1 through 5. After the add-on performs a successful operation it makes the following call:

```

if (TecUtilMacroIsRecordingActive)
    TecUtilMacroRecordExtCommand("SUMCELLS", "1-5");

```

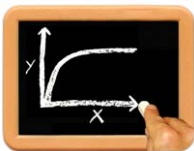
This will write out the following text to the macro file:

```

$!EXTENDEDCOMMAND
COMMANDPROCESSORID = "SUMCELLS"
COMMAND = "1-5"

```

`TecUtilMacroRecordExtCommand` requires that macro recording is active. This means that you must call `TecUtilMacroIsRecordingActive` and check the return value before calling `TecUtilMacroRecordExtCommand`.



Example 72: Adding Macro Commands to the Equate Add-on

Now we will examine code that allows *Equate's* compute function to be run from a macro command. All of the examples of the source code shown in this manual are included in the Tecplot 360 distribution in:

\$TEC_360_2013R1 /ADK/Samples/Equate. Refer to [“Equate Add-on”](#) on page 80 for the introductory steps for this add-on.

The first step is to decide what information is required by the add-on. In this case, *Equate* only requires that the value is sent to the Compute function. To write out the macro command, we will use the TecUtilMacroRecordExtCommand function. All TecUtil functions are defined in the [ADK Reference Manual](#).

Note the Compute_BTN_D1_CB function in *guich.c*:

```
static void Compute_BTN_D1_CB(void)
{
    char *strMulNum = NULL;

    TecUtilLockStart(AddOnID);
    strMulNum = TecGUITextFieldGetString(MulNum_TF_D1);

    if (TecUtilDataSetIsAvailable())
    {
        Compute(atof(strMulNum));

        if (TecUtilMacroIsRecordingActive())
            TecUtilMacroRecordExtCommand("equate", strMulNum);
    }
    else
        TecUtilDialogErrMsg("No data set available.");

    TecUtilStringDealloc(&strMulNum);

    TecUtilLockFinish(AddOnID);
}
```

We check to see if a macro is being recorded before we write out the macro command. When TecUtilMacroRecordExtCommand is called, it will add the following lines to the macro file:

```
$!EXTENDEDCOMMAND
COMMANDPROCESSORID='equate'
COMMAND='2'
```

COMMANDPROCESSORID tells Tecplot 360 which add-on to send the command to, and COMMAND is the value in the text field of *Equate's* dialog. Now that a macro command is being written out, write a function to decode it. When a macro is running, Tecplot 360 will send the information following COMMAND to the add-on. In this case, the only item that COMMAND contains is a number. Tecplot 360 sends all the information following COMMAND as a string.

Examine the following function in *main.c*:

In order to process macros, you must register a callback function. The callback must follow the signature

```
Boolean_t STDCALL ProcessEquateCommand(char *CommandString,
                                       char **ErrMsg)
{
    Boolean_t IsOk;

    TecUtilLockStart(AddOnID);

    IsOk = TecUtilDataSetIsAvailable();
    if (IsOk)
    {
        Compute(atof(CommandString));
    }
    else
    {
        *ErrMsg = TecUtilStringAlloc(2000, "Error message");
        strcpy(*ErrMsg, "No data set available.");
    }

    TecUtilLockFinish(AddOnID);

    return IsOk;
}
```

used as shown in the `ProcessEquateCommand`. This function is similar to the `Compute_BTN_D1_CB` function in `guicb.c`. There is no error checking of the value of `CommandString`. This is left as an exercise.

In `main.c` note the registration of the `ProcessEquateCommand` macro command callback from within the add-on initialization code:

```
EXPORTFROMADDON void STDCALL InitTecAddOn(void)
{
    TecUtilLockOn();

    AddOnID = TecUtilAddOnRegister(
        100,
        ADDON_NAME,
        "V"ADDON_VERSION("TecVersionId") "ADDON_DATE",
        "Joe Coder");

    if (TecUtilGetTecplotVersion() < MinTecplotVersionAllowed)
    {
        char buffer[256];
        sprintf(buffer, "Add-on \"%s\" requires Tecplot \"",
                "version %s or greater.",
                ADDON_NAME, TecVersionId);
        TecUtilDialogErrMsg(buffer);
    }
    else
    {
        InitTGB();

        TecUtilMenuAddOption("Tools",
                             "Equate",
                             'E',
                             MenuCB);
        TecUtilMacroAddCommandCallback("equate", ProcessEquateCommand);
    }

    TecUtilLockOff();
}
```

Equate is now complete. Compile and run your add-on. Try recording and playing back various macros to verify that the new functions you have added work properly.

Implementing Data Journaling

The initial loading of a data set and most changes made to a data set after loading are actions that can be journaled. In fact, most built-in data modification actions in the Tecplot Engine will automatically journal for you. However, custom data modifications performed directly by your application or add-on, must be journaled by hand. Executing the data journal recreates the data by loading data files and performing modifications. The figure below shows how the data journal plays an important role in the lifecycle of a layout file.

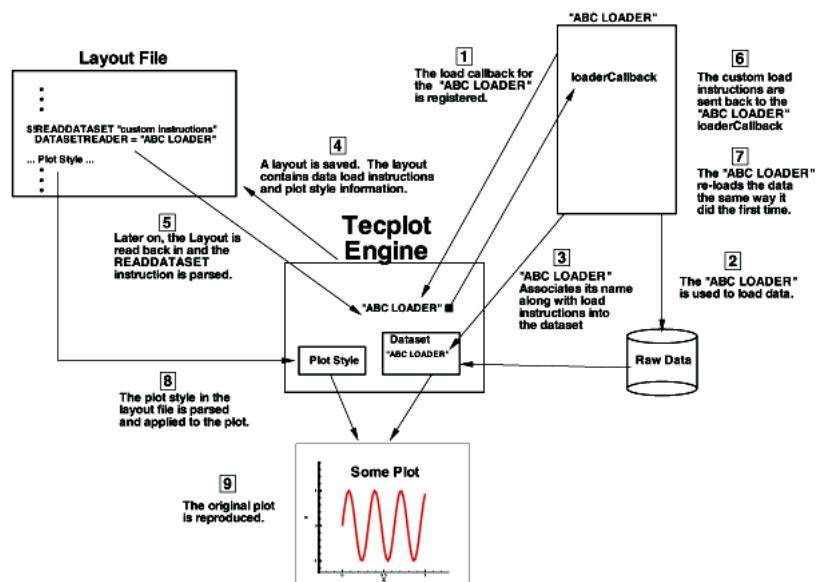


Figure 20-1. **The life cycle of a layout file.** Data journaling occurs in Step 6, when the data loader has finished loading and the load instructions have been registered.

When a layout is saved, rather than saving a new data file, the Tecplot Engine can reference the original data file and store only the modifications to the data within the layout file. This eliminates duplication of data and saves disk space.

20 - 1 Data Journaling Prerequisites

In order to journal a custom data modification, the following conditions must be met:

- There must be a data set attached to the current frame.
- The data set must have a valid journal.
- The data modification must involve only one data set.
- The data modification must not rely on style settings.

The first two prerequisites are easily verified with TecUtil functions calls. The third prerequisite arises from the fact that frames and their data sets can appear in layout files in any order. You cannot be certain that one data set will be created prior to another data set. As such, data journaling cannot rely on the order in which data sets are created. If your `your` creates or modifies one data set based on values in a different data set, this criterion is violated, and the operation cannot be journaled.

The final prerequisite arises from the fact that data journals in layout files are executed prior to any style commands. Style settings include axis, contour and other variable assignments. So if your `your` performs a custom data modification that uses the identity of the axis variables, then you need to include the axis variable assignments in your journal command, because your `your` will not be able to query the Tecplot Engine for that information when the journal is executed.

20 - 2 Inhibiting Marking of the Data Set

Unless your `your` inhibits it, any custom data modification it performs will “mark” the data set. Marking the data set invalidates the data journal and indicates to Tecplot that the data set must be saved to a new data file when a layout is saved. Therefore, to journal its actions your `your` must inhibit data set marking prior to performing its custom data modifications. Upon completion of its custom modifications, the `your` must re-enable marking. The `TecUtilDataSetSuspendMarking` function is used for both of these purposes.



Example 73: Data Journaling Code

The below code verifies there is a current data set. If so, it suspends data set marking and performs an action that modifies the data set. It then verifies that the data journal is valid, and records an `!EXTENDED_COMMAND` macro command to the data journal. Finally, it reactivates data set marking. Note that it stores the identity of the X-axis variable, which the `your` will need to use when the journal is executed:

```
if (TecUtilDataSetIsAvailable())
{
    EntIndex_t XAxisVar;
    char Command[256];
    /* Suspend dataset marking. */
    TecUtilDataSetSuspendMarking(TRUE);
    /* Modify the dataset. */
    ModifyDataSet();
    XAxisVar = TecUtilVarGetNumByAssignment(X);
    sprintf(
        Command,
        "DoStuff XAxisVar=%d",
        (int)XAxisVar);
    if (TecUtilDataSetJournalIsValid())
```

```
{
    TecUtilDataSetAddJournalCommand(COMMANDPROCESSORID,
                                    Command,
                                    NULL); /* RawData */
}

TecUtilDataSetSuspendMarking(FALSE);
}
```

Adding Online Help to Your Add-on

Once your add-on is complete, you may find that there are many details and instructions you would like to make available to users. Online Help is an effective way to include necessary details and instructions. It is an effective way of ensuring that needed information can easily be accessed from your add-on.

The Tecplot GUI Builder (TGB) provides an easy way for you to add on-line help to your add-on. Each modal and modeless dialog created with TGB has a [Help] button and associated callback. You are free to do any processing or use any help system you wish in the callback. Refer to [Chapter 25: “Tecplot GUI Builder”](#) for additional information.

The Tecplot 360 ADK API has also provided a function, `TecUtilHelp` which can display any HTML file when the user selects the [Help] button. This chapter will explain how to use `TecUtilHelp` to add on-line help to your add-on.

21 - 1 Step 1: Write your Help Pages

Write your help system as one or more HTML files. For simple add-ons, this may be a single HTML file. For add-ons that require more description, you may also use multiple files or an index page which references multiple files.

21 - 2 Step 2: Create a Help Directory

When you install your add-on, create a directory under the `$TEC_360_2013R1/help` directory. For example, if the name of your add-on is *MyAddOn*, create a directory: `$TEC_360_2013R1/help/MyAddOn/`. The name of the directory is not required to be the same as the name of your add-on.

21 - 3 Step 3: Processing the Help Button Callback

The TGB will generate a help button callback in *guicb.c* as follows:

```
static void Dialog1HelpButton_CB(void)
{
    TecUtilLockStart(COMMANDPROCESSORID);
    TecUtilDialogMessageBox("No help available");
}
```

```
TecUtilLockFinish(COMMANDPROCESSORID);  
}
```

Change this to:

```
static void Dialog1HelpButton_CB(void)  
{  
    TecUtilLockStart(COMMANDPROCESSORID);  
    TecUtilHelp("MyAddOn/file.html", FALSE, 0);  
    TecUtilLockFinish(COMMANDPROCESSORID);  
}
```

Here, *MyAddOn/file.html* represents the path of the HTML file to be displayed. Typically this will be named *index.html*, but this is not required. Note that by default, Tecplot 360 will prefix *\$TEC_360_2013R1/help/* to the file name, so you only need to specify the path below the Tecplot 360 Help directory. Alternatively, you can use an absolute path to your HTML file, such as *C:\MyDir\index.html*.



We strongly recommend you install your help files in Tecplot 360's help directory.

21 - 4 Using the Tecutilhelp Function

When an HTML (either a local HTML file or a valid URL) calls `TecUtilHelp`, the HTML file displays in the default browser installed on the system.

Windows platforms locate the default browser for you automatically using the system registry; you do not need to do any additional setup.

On Linux systems, you must specify the command which runs your browser by placing the following line in the Tecplot 360 configuration file (*tecplot.cfg*):

```
$!Interface UnixHelpBrowserCmd = <string>
```

where *<string>* is the command you would type at the command prompt to launch the browser. You must use the *@* symbol to mark the location to place the URL or filename, as shown in the following example:

```
$!Interface UnixHelpBrowserCmd = "\usr\bin\netscape @"
```

You may use any valid HTTP syntax for the HTML file name, as long the selected browser will accept it at the command line. For example, you may use special characters such as the octothorp (#) symbol to specify a particular location in the file.

For complete details on using `TecUtilHelp`, refer to the [ADK Reference Manual](#).



Example 74: Adding Help to the Equate Add-on

To add help to the *Equate* add-on, create a simple HTML document to serve as your help file, naming it *equate.html*.



Refer to [“Equate Add-on”](#) on page 80 and [“Adding Macro Commands to the Equate Add-on”](#) on page 201 for the preliminary steps to this tutorial.

In *guich.c*, note the call to launch the help in `Dialog1HelpButton_CB`:

```
static void Dialog1HelpButton_CB(void)
{
    TecUtilLockStart(AddOnID);
    TecUtilHelp("equate.html",FALSE,0);
    TecUtilLockFinish(AddOnID);
}
```

Place *equate.html* in the help subdirectory below the Tecplot 360 home directory, then recompile and reload your add-on. When you select Help on the **Equate** dialog, or press [F1] (Windows ONLY), *equate.html* will be launched in the default browser.

Working with Picked Objects

An *object* in the Tecplot Engine is any item that appears in the workspace that can be selected and can have actions performed on it. Geometries, axes, frames, legends, streamtraces, zones, and XY mappings are all examples of objects that can be picked.

There are a number of ways to select, or *pick*, objects. You can use `TecUtilMouseSetMode` to modify which actions are allowed after a pick has occurred. To programmatically pick objects, you can use `TecUtilPickAddAllInRect` to select objects within a confined rectangle, or `TecUtilPickAddAll` to select all objects of a specific type (e.g., all geometries or all zones). When an object is picked, *selection handles* are drawn (graphics that indicate that the object is selected and can be manipulated) or boxes around the object.

, picked objects are handled through the *pick list*. The pick list is an indexed list of objects that are currently selected. The objects in the list are typically sorted in the same order in which they were picked.

22 - 1 Object Types

Object types are used in many of the functions related to picked objects. An object type is specified with the `PickObjects_e` enumerated type. You may pick the following types of objects:

```
PickObject_Frame
PickObject_Axis
PickObject_3DOrientationAxis
PickObject_Geom
PickObject_Text
PickObject_ContourLegend
PickObject_ContourLabel
PickObject_ScatterLegend
PickObject_LineLegend
PickObject_ReferenceVector
PickObject_ReferenceScatterSymbol
PickObject_StreamtracePosition
PickObject_StreamtraceTermLine
PickObject_Paper
PickObject_Zone
PickObject_LineMapping
PickObject_StreamTraceCOB
PickObject_SliceCOB
PickObject_RGBLegend
PickObject_IsoSurfaceCOB
```

22 - 2 Picking Objects

To pick objects, you can use the `TecUtilSetMouseMode` function to set the mouse mode tool to be either the Selector or Adjustor. This will also clear out the pick list (i.e., unpick all picked objects).

The following functions are used to pick objects (i.e., to add objects to the pick list):

- `TecUtilPickAtPosition` - Pick an object at a specified (X,Y) location.
- `TecUtilPickAddAll` - Add all objects of a specified type to the pick list.
- `TecUtilPickAddAllInRect` - Add all objects of a specified type and within a specified region to the pick list.

The `TecUtilPickAtPosition` function can either replace or add to what is already in the pick list. The `TecUtilPickAddAll` and `TecUtilPickAddAllInRect` functions always add to what is already in the pick list. This makes it easy, for example, to pick all text and all geometries with the following two commands:

```
TecUtilPickAddAll(PickObject_Geom);  
TecUtilPickAddAll(PickObject_Text);
```

The functions `TecUtilPickGeom` and `TecUtilPickText` are also available to add a specific text or geometry to the pick list. The function that is used to unpick all objects or to clear out the pick list is `TecUtilPickDeselectAll`.

22 - 2.1 Picking Multiple Objects

In general, objects can only be selected within the current frame. This is true even with `TecUtilPickAddAllInRect` when the specified region encloses objects in other frames. For example, the following call will pick all zones within the current frame:

```
TecUtilPickAddAll(PickObject_Zone);
```

An object type of `PickObject_Frame` allows multiple frames to be selected.

Picking objects with `TecUtilPickAtPosition` can change the current frame, but only if objects are not being collected.

22 - 3 Operating on Picked Objects

Once you are satisfied with the currently picked objects, you can use the following functions to operate on the pick list:

- `TecUtilPickEdit` - Perform an action on all currently picked objects.
- `TecUtilPickCut` - Copy all currently picked objects to the paste buffer and then clear them from the plot.
- `TecUtilPickCopy` - Copy all currently picked objects to the paste buffer.
- `TecUtilPickPaste` - Paste all objects which are currently in the paste buffer to the plot.
- `TecUtilPickClear` - Clear all currently picked objects from the plot.
- `TecUtilPickShift` - Move all currently picked objects in the plot.
- `TecUtilPickMagnify` - Grow or shrink the size of all currently picked objects.
- `TecUtilPickPush` - Push all currently picked objects to the back of the plot (so that they are drawn earlier).
- `TecUtilPickPop` - Pop all currently picked objects to the front of the plot (so that they are drawn later).

Some of the above functions can only be used on specific types of objects. For example, `TecUtilPickMagnify` can only be used on frames, text, and geometries.



Example 75: Editing All Objects in the Pick List

The following code will add all geometries to the pick list and change their color, line pattern, position, and size:

```
TecUtilPickSetMouseMode(Mouse_Select);
TecUtilPickAddAll(PickObject_Geom);
TecUtilPickEdit("Color = Blue");
TecUtilPickEdit("LinePattern = Dashed");
TecUtilPickShift(1.2, 1.2, PointerStyle_AllDirections);
TecUtilPickMagnify(1.5);
```

22 - 4 Pick List

The pick list is accessed through index values starting at one. The procedure for enumerating the pick list begins with a call to `TecUtilPickListGetCount` to determine the number of objects that are currently in the pick list. This number can be used as a boundary condition as you loop through the pick list. For each object in the pick list, call `TecUtilPickListGetType`. This will return the type of object in the pick list at the specified index. Once you have the object type, you can call appropriate functions to get more information:

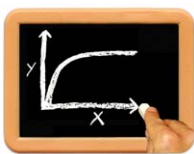
Function Name	Used Only For Object Type
<code>TecUtilPickListGetFrameName</code>	<code>PickObject_Frame</code>
<code>TecUtilPickListGetAxisKind</code>	<code>PickObject_Axis</code>
<code>TecUtilPickListGetAxisNumber</code>	<code>PickObject_Axis</code>
<code>TecUtilPickListGetZoneNumber</code>	<code>PickObject_Zone</code>
<code>TecUtilPickListGetZoneIndices</code>	<code>PickObject_Zone</code>
<code>TecUtilPickListGetLineMapNumber</code>	<code>PickObject_LineMapping</code>
<code>TecUtilPickListGetLineMapIndex</code>	<code>PickObject_LineMapping</code>
<code>TecUtilPickListGetText</code>	<code>PickObject_Text</code>
<code>TecUtilPickListGetGeom</code>	<code>PickObject_Geom</code>
<code>TecUtilPickListGetGeomInfo</code>	<code>PickObject_Geom</code>



Example 76: Changing the Color of Text and Geometries in Pick List

The following piece of code will change the color of all text and geometries in the pick list to purple:

```
int Index;
int Count = TecUtilPickListGetCount;
for (Index = 1; Index <= Count; Index++)
{
    PickObjects_e ObjectType = TecUtilPickListGetType(Index);
    /* We are only interested in text and geometries in the pick
list.*/
    switch (ObjectType)
    {
        case PickObject_Text :
        {
            Text_ID TextObject =
TecUtilPickListGetText(Index);
            TecUtilTextSetColor(TextObject, Purple_C);
        } break;
        case PickObject_Geom :
        {
            Geom_ID GeomObject =
TecUtilPickListGetGeom(Index);
            TecUtilGeomSetColor(GeomObject, Purple_C);
        } break;
    }
}
```



Example 77: Changing Color of Vectors in Pick List

The following piece of code will change the color of all vectors in the pick list to purple:

```
/* We will collect all the picked zones in a set. */
Set_pa Zones = TecUtilSetAlloc(TRUE);
int Index;
int Count = TecUtilPickListGetCount;
for (Index = 1; Index <= Count; Index++)
{
    PickObjects_e ObjectType = TecUtilPickListGetType(Index);
    /* We are only interested in the zones in the pick list. */
    if (ObjectType == PickObject_Zone)
    {
        EntIndex_t ZoneNum = TecUtilPickListGetZoneNumber(Index);
        TecUtilSetAddMember(Zones, ZoneNum, TRUE);
    }
}

TecUtilZoneSetVector("COLOR", Zones, 0.0, (ArbParam_t)Purple_C);
TecUtilSetDealloc(&Zones);
```

If instead we wanted to change the color of all objects in the pick list to purple, we could have used the following code:

```
TecUtilPickEdit("COLOR = PURPLE");
```

Part 6 User Interface

GUI Overview

You can incorporate a platform independent GUI into your add-on using the Tecplot GUI Builder (TGB). When you run `CreateNewAddon` (UNIX/Linux/Mac), you can choose to use the TGB during your add-on development. Windows users can create Windows-only dialogs by hand. Refer to [Chapter 25: “Tecplot GUI Builder”](#) and [Chapter 24: “Dialogs on Windows Platforms”](#) for detailed information.

23 - 1 UTF8 Strings

On Windows platforms, all TecGUI functions which take data types of `const char*` require valid UTF8 strings. If you pass a non-English UTF8 string to a TecGUI function (such as a Japanese UTF8 string), Tecplot 360 will display the string correctly (as long the proper language pack(s) have been installed on the your machine from the Windows XP® Installation CD).

Note: a valid UTF8 string is also a valid 7-bit ASCII string. As such, no changes are necessary if your existing add-on uses 7-bit ASCII strings exclusively (i.e., your strings use only ASCII characters in the range 0x00 - 0x7F).

IMPORTANT: If your add-on is currently using strings with ASCII characters in the range 0x80-0xFF to display accented European characters, the string will not display correctly in Tecplot 360 2013 R1. ASCII characters in this range are not standardized and are not valid UTF8, so you must convert your string to UTF8 (or to 7-bit ASCII) in order for it to display correctly.

23 - 2 Types of Dialogs

Tecplot 360 uses two primary types of dialogs - “modal” and “modeless.” *Modal dialogs* lock out the rest of Tecplot 360 from being used. *Modeless dialogs* do not. File dialogs, error messages, and the **Print** dialog are examples of modal dialogs. You must select [OK] or [Cancel] before interacting with the rest of Tecplot 360. The **Quick Edit** dialog, the **Rotate** dialog, and the **Zone Style/Mapping Style** dialogs are all examples of modeless dialogs. You may interact with the rest of Tecplot 360 while these dialogs are open.

A sidebar is a special subset of a modeless dialogs. Designing a sidebar is similar to designing a dialog, although a few different principles apply. Notably, sidebar dialogs can only be created using the Tecplot GUI Builder and you may only have one active sidebar at a time in Tecplot 360. Refer to [Section 25 - 2.2 “Adding Sidebars”](#) for additional information.

Tecplot 360's modal/modeless dialog paradigm does not exactly match that used on Windows operating systems, so add-ons must inform Tecplot 360 when a modal dialog is launched and dismissed. This allows Tecplot 360 (and other add-ons) to disable its interface. Add-ons with modeless dialogs must keep track of when to enable and disable their modeless dialogs. This is easily done by using Tecplot 360's state change mechanism (see [Chapter 6: "State Changes From an Add-on"](#)). Modal dialogs must inform Tecplot 360 (and other add-ons) of their launch/dismissal with `TecUtilStateChanged`.



This is done for you automatically if you are using TGB for your interface. The `BuildDialogn` function will call `TecGUIDialogCreateModal`.

Dialogs on Windows Platforms



This section is necessary only if you did not use the Tecplot GUI Builder while creating your add-on.

Tecplot 360 uses two types of dialogs - “modal” and “modeless.” *Modal dialogs* lock out the rest of Tecplot 360 from being used. *Modeless dialogs* do not. Examples of modal dialogs are file dialogs, error messages, and the Print dialog. All of these dialogs require you to select [OK] or [Cancel] before doing anything else inside Tecplot 360. Examples of modeless dialogs are the **Quick Edit** dialog, the **Rotate** dialog, and the **Zone Style/Mapping Style** dialogs. You may have these dialogs up and still interact with the rest of Tecplot 360.

This modal/modeless dialog paradigm does not exactly match that used by Windows operating systems, so add-ons must inform Tecplot 360 when a modal dialog is launched and dismissed so that Tecplot 360 (and other add-ons) can disable its interface. Add-ons with modeless dialogs must keep track of when to enable and disable their modeless dialogs. This is easily done by using Tecplot 360's state change mechanism. Modal dialogs must inform Tecplot 360 (and other add-ons) of their launch/dismissal with `TecUtilStateChanged`. Add-ons with modeless dialogs must monitor Tecplot 360's state with `TecUtilStateChangeAddCallback` and disable the modeless dialogs at the appropriate times.



This is done for you automatically if you are using TGB for your interface. The `BuildDialogn` function will call `TecGUIDialogCreateModal`.

24 - 1 Modal Dialogs

Whenever a modal dialog is launched, the add-on must inform Tecplot 360 of the launch by calling:

```
TecUtilStateChanged(StateChange_ModalDialogLaunch, NULL);
```

when processing `WM_INITDIALOG` or (in MFC) in the dialog's `OnInitDialog`.

Whenever a modal dialog is closed, the add-on must inform Tecplot 360 by calling:

```
TecUtilStateChanged(StateChange_ModalDialogDismiss, NULL);
```

when processing WM_NCDESTROY or (in MFC) in the dialog's PostNcDestroy.

That is all that is needed for modal dialogs. If you do not call these functions, Tecplot 360 may appear to work, but other add-ons may fail to work with your add-on.



You must call these functions in balanced pairs.

Dialogs created by Tecplot 360 (e.g. error messages, file dialogs) or any function starting with "TecUtilDialog" already process the state change.

24 - 2 Modeless Dialogs



The following section applies only if you are NOT using TGB for your interface.

If your add-on uses modeless dialogs like Tecplot 360 does, you have a little more work to do. First, you must monitor Tecplot 360's state changes. You do this by creating a state change callback function (see [Chapter 6: "State Changes From an Add-on"](#)). Within the state change callback, you must put the following:

```
{
/* When using MFC, the following line is always required on callbacks.
*/
/* Do not include this line if you are not using MFC. */
AFX_MANAGE_STATE(AfxGetStaticModuleState( ));

static int nDisabledCount = 0;

if (StateChange == StateChange_ModalDialogLaunch)
{
    nDisabledCount++;
    if (nDisabledCount > 0)
    {
        /* Disable all modeless dialogs, e.g., in Win32 SDK. */
        if (hwndModelessDialog)
            EnableWindow(hwndModelessDialog, FALSE);
        /* Or, under MFC. */
        if (modeless_dlg)
            modeless_dlg->EnableWindow(FALSE);
    }
}
else if (StateChange == StateChange_ModalDialogDismiss)
{
    nDisabledCount--;
    if (nDisabledCount <= 0)
    {
        nDisabledCount = 0;
        /* Enable all modeless dialogs, e.g. in Win32 SDK.
*/
        if (hwndModelessDialog)
            EnableWindow(hwndModelessDialog, TRUE);
        /* Or, under MFC. */
        if (modeless_dlg)
            modeless_dlg->EnableWindow(TRUE);
    }
}
```

```

    }
}

```

Each modeless dialog must be individually enabled or disabled. If you have several modeless dialogs, you should write a function to enable or disable all modeless dialogs.

If you do not monitor Tecplot 360's state, your add-on's modeless dialogs will not disable themselves at the appropriate times, and the user may be able to access your add-on at an unexpected time.



This could cause any number of serious problems, including a crash of Tecplot 360.

If you use both modal and modeless dialogs in your add-on, monitoring state changes for the modeless dialogs, and informing Tecplot 360 of the launch/dismiss of the modal dialogs, will automatically make your add-on's modeless dialogs disable themselves when the add-on itself brings up a modal dialog.



If you use the Tecplot GUI Builder for your interface, then the TecGUI function calls with enable or disable your Add-on dialogs.

24 - 3 PreTranslateMessage Function for Modeless Dialogs

In order to get keyboard navigation of your modeless dialog working on Windows platforms, you must add a PreTranslateMessage function and inform Tecplot 360 of this function. If you do not do this, the tab key, the escape key, the return key, and other keys will not work in your dialog.

Under MFC, the PreTranslateMessage function almost always looks as follows:

```

Boolean_t STDCALL PreTranslateMessage(MSG *pMsg)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return AfxGetApp->PreTranslateMessage(pMsg);
}

```

This will handle any number of modeless dialogs in your add-on.

Under Win32 SDK, the function should look as follows:

```

HWND hwndModelessDialog1; /* a modeless dialog */
HWND hwndModelessDialog2; /* another modeless dialog */

Boolean_t STDCALL PreTranslateMessage(MSG *pMsg)
{
    Boolean_t Result = FALSE;
    if (hwndModelessDialog1 != NULL)
        Result = IsDialogMessage(hwndModelessDialog1);
    if (!Result && hwndModelessDialog2 != NULL)
        Result = IsDialogMessage(hwndModelessDialog2);
    return Result;
}

```

If you have more than one modeless dialog in SDK, you need to call IsDialogMessage on each modeless dialog as long as IsDialogMessage does not return TRUE.

The TecUtil function to inform Tecplot 360 of your PreTranslateMessage function is TecUtilInterfaceWinAddPreMsgFn. You will usually call this function in your InitTecAddOn function:

```
EXPORTFROMADDON void STDCALL InitTecAddOn(void)
{
    /* When using MFC, the following line is always required on callbacks.
    */
    AFX_MANAGE_STATE(AfxGetStaticModuleState( ));

    TecUtilLockOn();
    COMMANDPROCESSORID = TecUtilAddOnRegister(110,
                                              "MyAddOn",
                                              "1.0",
                                              "Acme, Inc.");

    /* When using modeless dialogs, you must include a StateChange
    callback. */
    TecUtilStateChangeAddCallback(StateChangeCallback);

    /* And a PreTranslateMessage function. */
    TecUtilInterfaceWinAddPreMsgFn(PreTranslateMessage);

    /* Now, perform the rest of initialization. */

    TecUtilLockOff();
}
```

Tecplot GUI Builder

The Tecplot GUI Builder (TGB) is an add-on tool and library for building a platform-independent graphical user interface for a Tecplot 360 add-on. Using TGB will allow you to quickly generate platform-independent user-interface code in C, C++, or FORTRAN. However, it is not necessary to use TGB—you can use other commercial graphical layout tools, such as MFC.

When you run `CreateNewAddOn` (Linux/Macintosh OS® X) to begin your add-on development, a default set of TGB files are created. (See [Chapter 2: “Creating Add-ons on Linux/Macintosh Platforms”](#) for more details on using this utility.) This default code will generate a Tecplot 360 layout file called *gui.lay*. By default, *gui.lay* includes a blank dialog, which you can edit directly in Tecplot 360 using the **GUI Builder** dialog. The **GUI Builder** dialog is accessible from the **Tools** menu when you have included the TGB add-on in your *tecplot.add*¹ file. Visual Studio is not involved in editing or maintaining **GUI Builder** dialog layouts.

25 - 1 Using Tecplot GUI Builder

TGB is an add-on which generates the C, C++, or FORTRAN source code used to create dialogs and controls (e.g. push buttons and text fields).

Dialogs are laid out by creating frames in Tecplot 360 and then adding text and geometries to the frames. TGB distinguishes among different controls based on the style of the text and the keywords in the text. TGB's controls allow you to place new controls in a dialog easily, without requiring you to remember the particular text style for each type of control.

If you have enabled TGB from your *tecplot.add*¹ file, the Tecplot GUI Builder option will be accessible via the **Tools** menu in Tecplot 360.

25 - 1.1 How TGB Works

[Figure 25-1](#) depicts the main steps in building a graphical user interface for your add-on using TGB. These steps are:

1. To run Tecplot 360 with the TGB add-on, add **\$!LoadAddOn "guibld"** to your *tecplot.add* file. See also: [Chapter 4: “Running Tecplot 360 with Add-ons”](#).

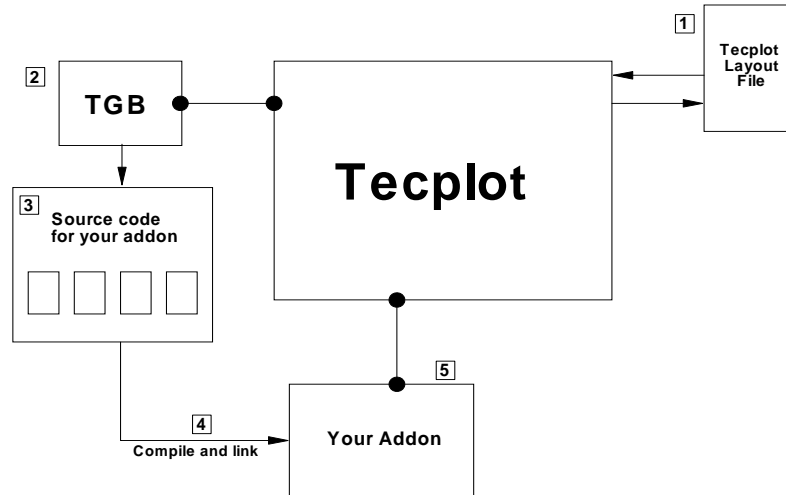



Figure 25-1. Building a graphical user interface using TGB.

1. Create or open an existing layout file that stores the information needed to define dialogs and the controls that go into the dialogs. If you used `CreateNewAddon`, a dialog template (*gui.lay*) will be included in your add-on development directory.
2. Open the **GUI Builder** dialog (accessed via Tecplot 360's **Tools** menu).
3. Using the **GUI Builder** dialog, add dialogs and controls to the layout file. Each dialog will be housed in a separate Tecplot 360 frame. Complete details of each of the available controls are included in the remainder of this chapter.
4. Once your dialog(s) are complete, select the  button from the **GUI Builder** dialog. This will overwrite *guicb.tmp* in your add-on development directory.
5. Copy the new code from *guicb.tmp* to *guicb.c*. Refer to [Section 25 - 5 "Modifying Your Source Code"](#) for additional information.



Add *guicb.tmp* to your development project, for easy access.

6. Modify the source code files as desired.
7. Compile and link the source code to create a shared library add-on.
8. Setup Tecplot 360 to run with your new add-on. Refer to [Chapter 4: "Running Tecplot 360 with Add-ons"](#) for additional information.
9. Your add-on will be loaded when you next run Tecplot 360.

Repeat the preceding steps as needed to make modifications to the graphical user interface for your add-on. The rest of this chapter describes each step in detail.

25 - 1.2 Selecting a Language

To select a language, go to **Layout>Options** in the **GUI Builder** dialog. You may select C, C++, or FORTRAN. The only difference between C and C++ is that C++ will generate files with the .cpp extension.

Internally, the files are identical with the C files. In the remainder of this document, reference to `.c/.cpp` files are interchangeable.

25 - 2 Adding Dialogs and Sidebars



If you used the `CreateNewAddOn` shell script to create your add-on, there will already be a default `gui.lay` file in your add-on directory, along with a number of default source code files. You are now ready to modify and/or extend the default GUI.



The following sections describe how to create and add controls to the dialogs. Before you add dialogs or controls to your GUI, you must first start Tecplot 360 and open the layout file that defines your GUI.

Each dialog, sidebar, tab and form will be represented as a separate frame.

25 - 2.1 Adding Dialogs

Dialogs are added by choosing either the Modeless Dialog  or Modal Dialog  buttons from the **GUI Builder** dialog. Each dialog is created in a new frame in Tecplot 360. Information about the dialog is incorporated into the Frame Name using a collection of keywords. You can edit the frame name in the **Edit Active Frame** dialog (accessed in the **Frame** menu). Refer to the following table for a description of the available keywords.

Keyword	Default	Description
ID= <i>n</i>	Dynamic (TGB automatically generates a unique number)	The dialog ID is assigned to be <i>n</i> . Do not change this once you start generating source code.
TITLE= <i>string</i>	"Untitled"	The title at the top of the dialog.
MODE= <i>mode</i>	Dynamic (based on the selection in the GUI Builder)	MODE can be either "MODAL" or "MODELESS".
OKCLOSEBUTTON= <i>boolean</i>	TRUE	Includes an okay/close button in the dialog.
CANCELBUTTON= <i>boolean</i>	TRUE	Includes a cancel button in the dialog (applies to modal dialogs only).
HELPBUTTON= <i>boolean</i>	TRUE	Includes a help button in the dialog.
APPLYBUTTON= <i>boolean</i>	FALSE	Includes an apply button in the dialog.

In addition to the keywords, the frame title may also begin with a comment enclosed in square brackets. This is useful if you have a lot of dialogs and want to quickly identify and pop them using Tecplot 360's frame ordering function. For example, the frame title could be:

```
[This is the main dialog] ID=1 TITLE="Main Dialog" MODE=MODAL
```

25 - 2.2 Adding Sidebars



The *sidebartest* example add-on illustrates the TGB sidebar API described following.

Your add-on can use its own *sidebar dialog* in lieu of the standard Tecplot 360 sidebar. The sidebar may contain any controls available in the TGB, with the exception of menus. All controls in a sidebar receive the same callbacks that are used for a modal or modeless dialog.

An add-on may create several sidebar dialogs and switch between them using `TecGUISidebarActivate`. The Tecplot 360 sidebar may also be restored by calling `TecGUISidebarActivate` with the special ID value of `TECGUITECPLOTSIDEBAR`.




Only one sidebar dialog may be displayed at a time.

Add-ons may create sidebars which are wider or narrower than the Tecplot 360 sidebar. The handling of different sized sidebars may be changed with the following macro command:

```
$!Interface SidebarSizing = {Dynamic | MaxOfAll}
```

If `SidebarSizing` is set to `MaxOfAll` (default), the sidebar size will be set to the maximum width of all registered sidebars, including Tecplot 360. If **SidebarSizing** is set to `Dynamic`, the sidebar size will be adjusted each time a new sidebar is displayed.

Creating sidebars

Designing a sidebar in the TGB is the same as designing a dialog. To create a sidebar in TGB, select the Sidebar button  in the **GUI Builder** dialog. A dialog will be created in the TGB with the approximate width and height of the Tecplot 360 sidebar. The frame title will have the keyword `SIDEBAR=TRUE`.

After you have created the sidebar in the TGB, you may add controls just as you would a modal or modeless dialog. TGB will generate the code to create the sidebar and add controls to it in a function `BuildSidebarNN`, located in *guibld.c*. This function is automatically called when you call the generated `BuildDialogN` function.

Activating and deactivating sidebars

Only one sidebar can be active at a given time. You can switch sidebars through the **Options** menu. When you select a different sidebar from the menu, the current sidebar will be replaced.

After calling `BuildDialogN`, you can activate it by calling `TecGUISidebarActivate(SidebarNN)`. You can activate the Tecplot 360 sidebar by calling `TecGUISidebarActivate(TECGUITECPLOTSIDEBAR)`.

To remove all sidebars including Tecplot 360's, call `TecGUISidebarDeactivateAll`. The user can deactivate all sidebars by using the **None** option in the **Options** menu.

25 - 3 Adding Controls

Controls are added to dialogs and sidebars via the controls buttons in TGB. The TGB allows you to add the following types of controls to your sidebar or dialog:

- [Bitmap Buttons and Toggles](#)
- [Push Buttons](#)

- [Scale Controls](#)
- [Form Controls](#)
- [Tab Controls](#)
- [Text Field Spin Controls](#)
- [Option Menus](#)
- [Adding a Menu Bar to a Dialog](#)

25 - 3.1 Common Control Features

The following principles apply to most of the controls you can add to your dialog or the general layout of your dialog:

- [Variable names](#)
- [ToolTips](#)
- [Group boxes and separators](#)
- [Anchor controls to paper](#)
- [Alignment options](#)
- [Previewing dialogs](#)

Variable names

Controls which generate events must have a unique variable name. The variable name is specified in the "Macro Command" text field of the **Text Options** dialog using the VARNAME= keyword. To change the "Macro command" text field, select the [Options] button in the **Text Details** dialog.

For example, to assign the variable name "banana" to a toggle, in the "Macro Command" field, change the default VarName = TGL to varname=banana.



Because variable names may not contain spaces, double quotes are *not* used.

Control keywords

If a name is not assigned as the Macro Function Command, TGB assigns a name by looking at the text used for the control. Some controls must begin with a keyword identifying the control type. [Table 25 - 1](#) lists the controls, the keyword, and text style that TGB uses to identify the control type:

Control (X Motif)	Control (Windows)	Keyword	Tecplot text style
Label	Static text	None	Plain text (no text box)
Form	Form	FM:Group=NN	Filled text box
Multi-line text field	Multi-line edit	T:	Filled text box (multi-line)
Multi-selection list	Multi-selection list box	MLST:	Filled text box
Option menu	Combo box (drop-down menu)	OPT:	Filled text box

Table 25 - 1: Control Keywords

Control (X Motif)	Control (Windows)	Keyword	Tecplot text style
Push button	Push button	None	Filled text box
Radio box	Radio box	$\langle \text{math} \rangle 7 \langle / \text{math} \rangle$	Hollow text box
Read only multi-line text field	Multi-line read only	TRO:	Filled text box
Scale	Slider	SC:	Filled text box
Set of tabs	Property sheet	TAB:Group=NN	Filled text box
Single-selection list	Single-selection list	SLIST:	Filled text box
Text field	Edit	TF:	Filled text box
Text field with spin	Text field with spin	TFS:	Filled text box
Toggle	Toggle	$\langle \text{math} \rangle 7 \langle / \text{math} \rangle$	Plain text
Bitmap Button	Bitmap Button	None	Image geometry
Bitmap Toggle	Bitmap Toggle	None	Image geometry

Table 25 - 1: Control Keywords

For all controls (except labels and push buttons), TGB can use the text after the keyword to determine a name for the control. This is only used if the Macro Function Command field, mentioned earlier, is not used. In order for TGB to identify the control type correctly, the keywords from [Table 25 - 1](#) must be used at the beginning of the actual text used for the control. TGB does not look for these keywords in the Macro Function Control field.



These keywords are created automatically when you use the **GUI Builder** dialog to create the control.

For example, if you have a toggle with the text $\langle \text{math} \rangle 7 \langle / \text{math} \rangle$ Include Banana, TGB will name the control "Include Banana." Note that $\langle \text{math} \rangle 7 \langle / \text{math} \rangle$ signifies toggles and the radio box because it resembles a check box when it is displayed on the screen. For labels, TGB uses the label text. By default, variable names are limited to 30 characters. To remove this limitation, select **Layout>Options** in the **GUI Builder** dialog and deselect the "Limit variable name length to 30 characters" toggle. Note that if you deselect this option on an existing add-on project, you may have to edit your *guib.c* file before compiling your add-on, since the variable names may change with the longer length limit.



You cannot generate variable names with more than thirty characters in FORTRAN.

ToolTips

You can add a tool tip to any control by using the Tool Tip keyword in the "Macro Command" field. For example, to add a tool tip to the "banana" variable, type the following:

```
varname=banana tooltip="This is the banana toggle"
```

Note that since a tool tip is a string, double quotes are *required*. You may add a tool tip to any type of TGB control. Alternatively, you can add a tooltip by calling `TecGUISetToolTip` with the ID of the control.




IMPORTANT: In previous versions of TGB, you could specify a variable name by typing it into the "Macro Command" text field. This syntax is still supported. However, if you want to use a tool tip, you *must* use the `VARNAME=` identifier. This allows the TGB to differentiate between the variable name and the tool tip text.

Group boxes and separators

Group boxes are rectangles that surround groups of controls in a dialog. A group box can be added by adding a rectangle geometry to a frame in Tecplot 360. In addition, you can add a label to the group box. This is done by adding the text for the label into the Macro Function field for the rectangle geometry.

Horizontal and Vertical separators can also be added by creating a simple two point line segment geometry that is either horizontal or vertical. Note that you can press the "H" or "V" keys on the keyboard while drawing the line segment and Tecplot 360 will force it to be horizontal or vertical, respectively.


Anchor controls to paper

When you select the Anchor button , all of the controls will be anchored to the paper. This allows you to resize a dialog without changing the position of any controls. Note that you should select this button only when you are resizing the dialog. Otherwise, it should be unselected.

Alignment options

To align controls using the alignment options, select 2 or more controls and press the appropriate alignment button. Note that, when appropriate, the first control selected controls the alignment. For example, selecting the **Align Left** button will align all of the controls based on the left margin of the first selected control.

Previewing dialogs

You may preview a dialog by selecting the dialog and selecting the Preview button  from the **GUI Builder** dialog. Previewing is useful for checking the layout of a dialog before generating the source code. Only one dialog may be previewed at a time. Use the [Close], [OK] or [Cancel] button in the previewed dialog to exit preview mode.



(Linux Only) Each time you preview a dialog, that dialog's resources are created from scratch. Since dialog resources are not released until you exit Tecplot 360, excessive previewing during the same Tecplot 360 session will gradually reduce the available resources. When designing a dialog in TGB, we recommend that you periodically close and restart Tecplot 360 if you are frequently previewing dialogs.

25 - 3.2 Bitmap Buttons and Toggles



A "Bitmap button" is a button which contain a bitmap rather than text. Bitmap toggles are identical to bitmap buttons, except that they will stay "pushed" when selected, with the pushed state representing "toggle on". Bitmap controls have two sizes: the dimensions of the control itself and the pixel dimensions of the bitmap to be placed in the control.

When you run an add-on, Tecplot 360 will create a bitmap control as follows:

1. The control (button or toggle) will be created using the control dimensions.
2. A bitmap will be created using the pixel bitmap dimensions and centered in the control.

Note that the bitmap is never resized to fit onto the control. If the bitmap is too large, then it will be clipped to the size of the control. If it is too small, then there will be additional empty space on the control.

When you add a bitmap control in TGB, TGB will create an image geometry of the appropriate size for the bitmap dimensions. Resizing the control is not recommended. You may resize this button, but note that the original bitmap size does not change and will be centered on the button, or clipped if it is too large.

You create bitmap buttons or bitmap toggles in TGB by selecting **bitmap button**  or **bitmap toggle** , respectively. After you select one of the bitmap control buttons, you will be prompted to specify the bitmap file name. Note that this file is not needed when you run your add-on. It is only needed when TGB generates source code for the add-on. TGB will generate a static array of bytes in the *guicb.c* file which represent the bitmap. The bitmap is generated in true color (24bit RGB) format.

After you have specified the bitmap file name, the **Insert Bitmap Image** dialog will be launched. Specify the following options on the dialog:


- **Tool tip text** - We recommend that you add Tool tip text to your bitmap controls, since it is not always clear to users what a bitmap button does. The text you enter in the "tool tip" field will be added to "Macro Command" text field of the **Image Geometry Details** dialog in the form:

Tooltip = "Tool Tip Text"

You may edit the tool tip at any time.

- **Button Type** - Select a bitmap button or bitmap toggle
- **Use transparent color** - Most bitmaps have a transparent color. Check this toggle to enable the bitmap to have a transparent color. The transparent color will be replaced by the button background color when you run your add-on.
 - **Transparent color** - Enter the transparent color in as RGB in 6 digit hexadecimal format (similar to HTML). Examples: Red: FF0000 Black: 000000, White: FFFFFFFF, Blue: 0000FF

25 - 3.3 Push Buttons


Select the  to add a push button control to your dialog. Add a text label to your control by double-clicking on the new button and editing the text in the **Text Details** dialog.



Push button controls are sized to their label text, by default. If you plan to add multiple push buttons to your dialog and would like them to be the same size, follow these steps:

1. Use the same text in all of your push buttons while constructing your dialog/sidebar.
2. Once you have built the GUI, use `TecGUIButtonSetText` during dialog initialization to set the desired text labels in *guicb.c*.

25 - 3.4 Scale Controls

To add a scale control to your dialog, select the  button from the **GUI Builder** dialog. Use the `TecGUIScale` commands to specify the scale characteristics, including limits and precision. Scale controls have their numeric labels hidden by default. If you would like the label to be visible by default, use the macro command keyword: `SHOWLABEL=TRUE` in the **Text Options** dialog.


25 - 3.5 Form Controls


Forms are typically used to display different sets of options based on selections made by the user. A *parent* form control is a rectangular region of a dialog that can show and hide different sets of controls at different times. A *child* form page is a set of dialog controls which can be shown or hidden inside the parent form control. Child form pages are shown as separate dialogs in TGB, while the parent is displayed with the dialog. While you are working with the child form dialogs in TGB, the pages will have a cyan background in order to distinguish them from normal dialogs.

The primary difference between child form pages and regular dialogs is that the size of a form page dialog is restricted to the size of its parent form control. You may add any number of form pages to a form control. Controls are added to form pages and built by TGB exactly like dialogs.



An add-on can have no more than twenty form or tab controls. This is because forms and tabs make use of frame-linking, which is limited to twenty distinct linking groups. Each form or tab may have an unlimited number of pages, however.

To create a new form in a **GUI Builder** dialog, select the Parent Form button . This will add a new control to the dialog with the type FM:Group=NN, where NN is an automatically generated link group number. *Do not edit this text or the group number!* It is needed by TGB to identify the control as a form and to identify the form pages associated with this control. When you add a new form to a dialog it is initially empty. In order to create sets of controls, you must add child form pages.

To add a form page, select the Form Page button . *The button is active only if you have selected a form control.* This will create a new dialog in TGB which is actually a form page. This new dialog (a Tecplot 360 frame) will have the same link group number as the Group=NN text in its parent form control. It will also be exactly the same size as the form control it is linked to. *Do not edit or remove the group linking from this frame,* or it will not be recognized by TGB as a form page. In the frame name will be an additional keyword, FORMPAGE=T, which identifies this dialog as a form page. See [Section 25 - 3 "Adding Controls"](#) for a complete list of keywords for form and tab pages.

In your source code, TGB will generate a variable representing each form page using the parent form. This variable is similar to the DialogNManager variable that TGB creates for each dialog.



To set the controls for a form control to this set, call:

```
TecGUIFormSetCurrentPage(FormN_GManager)
```

Where G is the group number associated with the form and N is the form page number.

See also: [Section "Resizing parent form or tabs"](#) on page 234.

25 - 3.6 Tab Controls

Tabs are identical to forms except that tabs have an additional set of controls above the parent form area. The form is automatically changed for you when the user selects a tab. Individual tab pages are added using the Tab , and Tab Page  buttons on the **GUI Builder** dialog.

Tab pages have the additional keyword, POS=NN, in the frame name. This specifies the position from left to right. See [Section 25 - 3 "Adding Controls"](#) for a complete list of keywords. If you wish to reorder a set of tab pages, you may edit the POS=n value in the frame name. Note that the ordinal *n*'s do not have to be consecutive, since tab pages are sorted by TGB in ascending order before generating source code. For example:

```
POS=1, POS=8, POS=10
```

is equivalent to


```
POS=1, POS=2, POS=3
```

Each form and tab page frame must also have a linked group number, allowing TGB to associate a set of pages with a parent control. This is done automatically when you use Add Form, Add Tab, Add Form Page, or Add Tab Page buttons.

Keyword	Default	Description
FORMPAGE=boolean	FALSE	TRUE if this dialog is a form page
TABPAGE=boolean	FALSE	TRUE if this dialog is a tab page

Keyword	Default	Description
POS=n	None	If this dialog is a form page, the NN is its position, with NN=1 at the left
TITLE="string"	Page n	Title of the tab page

Resizing parent form or tabs

If you change the size of a parent form or tab control, you must select **Resize**  on the **GUI Builder** dialog. This resizes all of the linked forms or tab pages to reflect the new size of the parent form or tab. If you do not resize the tab or form pages to match the new size of the parent control they will not be sized correctly in the final GUI.

25 - 3.7 Text Field Spin Controls


In TGB a text field spin control is a text field with two small arrow buttons anchored at the right end of the text field control. Spin controls are interchangeable with text field controls, and may be passed to any TecGUI function requiring a text field control.

In addition to the text changed callback, spin controls also receive a callback when users select **up** or **down** arrows. It is up to the add-on to manage the text inside the control. This typically involves incrementing or decrementing a numeric value in the text control then re-displaying it. However, this is not a requirement. Spin controls may contain any text which may be changed in any way when up or down arrows are selected.



For a sample TGB Add-on which uses forms, tabs and spin controls, see the sample code for Tabtest. The ADK sample directory is included in your installation. It is placed in %\$TEC_360_2013R1%\Adk\Samples.

25 - 3.8 Option Menus

Use the Option Menu button  to incorporate an option menu into your dialog. The values of the options must be specified programmatically using the TecGUIOption functions.

Using C/C++

When generating interface code in C, TGB creates a static string in the guicb.tmp file that is used to store the options for the option menu. For example, if you have an option menu control with the name "Fruit" then guicb.tmp will contain the declaration:

```
static char *Fruit_OPT_D1_List = "Option 1,Option 2,Option 3";
```

After transferring this to guicb.c you can edit the string and put the items you want to appear in the option menu in the static string. Separate items with a comma. For example, the resulting declaration in guicb.c may appear as:

```
static char *Fruit_OPT_D1_List = "apple,banana,orange";
```

Using FORTRAN

Option menu coding in FORTRAN is different than C, because TGB does not give you any hints as to what to add. The procedure is the following:

- Find the spelling of the character string created to hold the option menu items. It will be located in the file GUIDefs.INC. For example, if an option menu to assign colors is named

coloropt and is in the first dialog, then you will find the variable coloropt_OPT_D1_List in GUIDEF5.INC. It is of type character*100 by default.

- Put all assignments for the character strings that define the option menus into the InitTecAddOn113 function. It is critical that this assignment is made at the very beginning. If the color choices are "red," "blue," and "green," then the statement to add to the initialization function is as follows:

```
Subroutine IntTecAddOn113
:
:
:
Call TecUtilLockOn
coloropt_OPT_D1_List = "Red,Blue,Green"
```

25 - 3.9 Adding a Menu Bar to a Dialog

Menu Bars cannot be added via the TGB, they must be added by hand. A menu bar is constructed as follows:

- Call **TecGUIMenuBarAdd**.
- For each menu option to add to the menu bar call **TecGUIMenuAdd** using the ID of the menu bar as the parent.
- For each menu item added to a menu option call **TecGUIMenuAddItem**.

Other **TecGUIMenu** functions are available to add items such as toggled menu items and to modify the menu structure once it is in place.



Also note that **TecGUIMenuAdd** can be used to create walking menus by using another menu as the parent instead of the menu bar.

The menu creation code must only be executed once and should be done so immediately after the creation of the dialog. The best place to put the code is immediately after the call to **BuildDialog** for the dialog. The following example demonstrates how to do this in a way that guarantees the menu bar code will only be executed once.

Create a menu bar that has the following menu structure:

```
Main Menu Bar
+--> File
    +--> New Project
    +--> Open Project
    +--> Save Project
+--> Setup
    +--> Solver Setup
    +--> Reference Values
    +--> Define Output
        +--> Print
        +--> Integration
        +--> History Plot
        +--> Solution Plot
... in the callback to launch the dialog...
if (Dialog1Manager == BADDIALOGID) /* i.e., the dialog has not been
built */
{
    BuildDialog1(MAINDIALOGID);
    MenuBar = TecGUIMenuBarAdd(Dialog1Manager);
    FileMenu = TecGUIMenuAdd(MenuBar,"File");
    NewProject_item = TecGUIMenuAddItem(FileMenu,"New Project",
                                        Add item to menu",
```

```

        NewProject_MN1_D1_CB);
OpenProject_item = TecGUIMenuAddItem(FileMenu,"Open Project",
        OpenProject_MN1_D1_CB);
SaveProject_item = TecGUIMenuAddItem(FileMenu,"Save Project",
        SaveProject_MN1_D1_CB);
SetupMenu = TecGUIMenuAdd(MenuBar,"Setup");
SolverSetup_item = TecGUIMenuAddItem(SetupMenu,
        "Solver Setup",
        SolverSetup_MN2_D1_CB);
ReferenceVal_item = TecGUIMenuAddItem(SetupMenu,
        "Reference Values",
        ReferenceVal_MN2_D1_CB);
DefineOutput_menu = TecGUIMenuAdd(SetupMenu,"Define Output");
PrintOutput_item = TecGUIMenuAddItem(DefineOutput_menu,
        "Print",
        PrintOutput_MN2_D1_CB);
Integration0_item = TecGUIMenuAddItem(DefineOutput_menu,
        "Integration",
        Integration0_MN2_D1_CB);
HistoryPlot0_item = TecGUIMenuAddItem(DefineOutput_menu,
        "History Plot",
        HistoryPlot0_MN2_D1_CB);
SolutionPlot_item = TecGUIMenuAddItem(DefineOutput_menu,
        "Solution Plot",
        SolutionPlot_MN2_D1_CB);
}
...

```

25 - 3.10 Specifying Exact Control Width in Dialog Units or Characters

By default, the width of a dialog control is determined by the width of the control in the dialog layout. However, in some cases you may need to specify an exact width for a dialog control. For example, you may need a text field spin control to be only 1 character width wide if it will only contain single digit numbers. However, in a dialog layout the minimum width for a text field spin control is 4 characters, since all such controls must have at least "TFS:" in them. [See [Section "Control keywords"](#) on page 229].

You can use the control keywords `UnitWidth` and `CharWidth` to override the layout width and specify an exact width for a TGB control as follows:

- **UnitWidth** - Use the keyword to specify the exact width in dialog units for the control.

`UnitWidth=n`, where $n > 0$

The value you enter here will be passed directly to the `TecGUI` function which creates the control. For example, in a button control, specifying:

`UnitWidth=4367`

will cause TGB to generate code as follows:

```
TecGUIButtonAdd(ID,X,Y,4367,Height,"Button",Callback);
```

- **CharacterWidth** - Use this keyword to specify the exact width in characters for the control.


`CharWidth=n`, where $n > 0$

For example, in a text field spin control,

`CharWidth=1`

will cause TGB to generate code such that the width of the spin control is exactly one character (suitable for single digits).

25 - 4 Building the Source Code

To build source code for your dialog(s) or sidebar(s), select the **Go Build**  button at the bottom of the **GUI Builder** dialog. TGB will generate the source code for your GUI and at the same time save the Tecplot 360 layout file so it reflects the changes you have made. You can now exit Tecplot 360, merge the generated GUI code with the previous GUI code, and compile your add-on.

When you build your dialog(s) and sidebar(s), the TGB saves your dialog as a layout file and creates the following files:

Description	Filename		
	C	C++	FORTRAN
Template for the callback module, contains code for each control in your dialog	<i>guicb.tmp</i>		
Interface builder module	<i>guibld.c</i>	<i>guibld.cpp</i>	<i>guibld.f</i>
Include file naming all of the controls	<i>guidefs.h</i>	<i>guidefs.h</i>	<i>guidefs.inc</i>
list of global variables	<i>guidefs.c</i>	<i>guidefs.cpp</i>	N/A
include file naming all of the callback functions	N/A	N/A	<i>guib.inc</i>

The file *guicb.tmp* is the template for the *guicb.c* (or *guicb.f*) module you will be editing to customize all of the callbacks generated by your interface. A callback is a function that is called when the user interacts with one of the controls in your dialogs. The first time you run TGB, simply rename *guicb.tmp* to *guicb.c* (or *guicb.f*). If you change and rebuild your layout later, you will need to cut and paste the corresponding code from *guicb.tmp* to *guicb.c/guicb.f*. Refer to [Section 25 - 5 “Modifying Your Source Code”](#) for additional information.



You should never edit the files *guibld.c*, *guibld.f*, *guidefs.c* or the include files, *GUIDEFS.h* (C language), *GUIDEFS.INC* and *GUICB.INC* (FORTRAN language). If you ever modify any of these files, be aware that they will be overwritten the next time you run Tecplot GUI Builder

25 - 5 Modifying Your Source Code

The file *guicb.c* (or *guicb.F* for FORTRAN) contains the functions called whenever a control in your interface is operated by the end user. For example, suppose you have a push button that is labeled “Eject.” TGB will then create code for a function called *Eject_BTN_CB_D1* that is called when the button is pressed. TGB names the functions according to the base string that you provide (“Eject” in this case, see [Section 25 - 2.2 “Adding Sidebars”](#)). It also adds information to uniquely identify the function. Here *BTN_CB* means this is a push button callback and *D1* means the button resides in dialog number 1.

If you later decide to make changes to the interface, and the changes involve more than the placement of controls or shape of the dialog, you must make changes to the *guicb.c* or *guicb.F* file.

For example, if you add a new push button to a dialog you would perform the following steps:

- Look at the *guicb.tmp* template file that is generated. It contains a new callback function for the new button.
- Cut and paste this new function from *guicb.tmp* to the existing *guicb.c* or *guicb.F* file. You can then add code to carry out the button press action.

If you remove a control from a dialog, it is not necessary to edit *guicb.c* or *guicb.F*. However, if you do not, you will end up with a callback function that is never called.

If you rename a control, you should look at *guicb.tmp* and see how TGB has now named things, then edit *guicb.c* or *guicb.F*. Change the name of the callback function to match.

25 - 6 Compiling Your Add-on

25 - 6.1 Linux and Macintosh OS X Platforms

Compiling the add-on consists of running the Runmake shell script provided in the distribution. You can run Runmake with no parameters, or you can add options for platform and type of executable on the command line. For example, if your platform is sgix.62, use:

```
Runmake sgix.62 -debug
```



Always use the `-debug` flag when developing add-ons. Only when you are ready to make a release version use the `-release` flag. Using `-debug` puts the resulting shared library in the appropriate location so that Tecplot 360 will know where to get it when using the `-develop` flag.

25 - 6.2 Windows Platforms

In Visual Studio, select the **Build** button or the **Rebuild** button. Refer to [Section 3 - 2 "Creating an Add-on with Visual Studio 2005"](#) on page 30 for additional information.

25 - 7 Informing Tecplot 360 of Your New Add-on

This step is only required if you are developing add-ons under Linux or Macintosh OS X.

If you have just created this TGB add-on, then you must inform Tecplot 360 of its existence by editing the *tecdev.add* file in the add-on development root directory and adding the entry, `!LoadAddOn "|TECADDONDEVDIR|/libmyaddon"`, where *myaddon* is the base name of your add-on.

See also: [Chapter 4: "Running Tecplot 360 with Add-ons"](#).

25 - 8 Running Your New Add-on

Linux platforms or Macintosh OS X

To run the debug version of your new add-on you must set the environment variables:

```
TECADDONDEVDIR=myaddondevdir
TECADDONDEVPLATFORM=myplatform
```

where:

- *myaddondevdir* is the path to the directory above your add-on projects. This is the directory from which you run `CreatNewAddOn`, to create your add-on in the first place. We recommend that you add the preceding environment variable values to your `.cshrc` or `.profile` files (or `.bashrc` or `.bash_profile` for bash shell).
- *myplatform* is the same platform name you used with Runmake.

After setting up these environment variables, run Tecplot 360 using:

```
tec360 -develop
```

Windows platforms

In Visual Studio, select [Go] or press F5.



Example 78: Extending Interactive User Interface Capabilities

The *SumProbe* add-on illustrates how to sum the probed values of a selected variable. It will appear in Tecplot 360's **Tools** menu as "Sum Probed Values". When selected, a dialog will appear allowing you to specify which variable you wish to sum.

SumProbe uses source code files created by the *CreateNewAddOn* script (Linux/Macintosh). All of the examples of the source code shown in this manual are included in the Tecplot 360 distribution in the *adk/samples/sumprobe* below the Tecplot 360 home directory.

Step 1 Create a new add-on

When running *CreateNewAddOn*, answer the questions as follows:

- **Project Name** - SumProbe
- **Add-on name** - Sum Probe
- **Company name** - Your company name]
- **Type of add-on** - General Purpose
- **Language** - C
- **Use TGB to create a platform-independent GUI?** - Yes
- **Add a menu callback to the Tecplot 360 "Tools" menu?** - Yes
- **Menu text** - Sum Probed Values
- **Menu callback option** - Launch a modeless dialog
- **Dialog Title** - Sum Probe

We will use a *TecUtil* function to get the variable name to sum and TGB to create a dialog to display the total number of summed points.

After running *CreateNewAddOn*, you have the following files:

<code>guibld.c</code>	<code>guicb.c</code>	<code>guidefs.c</code>	<code>main.c</code>
<code>ADDONGBL.h</code>	<code>GUIDEFS.h</code>	<code>gui.lay</code>	

You will also have other files specific to your platform, but we will only modify those listed. The purpose of each file will be explained in detail as we proceed.

Verify that you can compile your project add-on and load it into Tecplot 360. If you cannot, refer to [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) or [Chapter 3: "Creating Add-ons on Windows Platforms"](#).

Step 2 The MenuCallback function

Most add-ons contain a callback function named *MenuCallback*. This is called by Tecplot 360 each time the add-on is selected from the **Tools** menu. *MenuCallback* stores the code that performs all functions of the

add-on. This callback function is specified in the `TecUtilMenuAddOption` function is passed to Tecplot 360 in `InitTecAddOn`.

The `TecUtilDialogGetVariables` function has a built-in dialog which allows you to select the variable to be summed. Then the newly-created dialog appears. As points are probed, the summed total is displayed on the dialog.

Before adding the following code, create a label on the dialog which will be set to the total as the plots are probed. (See the [Chapter 25: "Tecplot GUI Builder"](#) for more information on adding a label to a **GUI Builder** dialog.) Set the variable name of this label to `VarName=Totalis00`. Set the text string of the label to read "The total is 0.0"

The new or modified source code is displayed in bulleted lines. If you are working along, add or edit bulleted lines only.

Note the `MenuCallback` function in `main.c`:

```
static void STDCALL MenuCallback(void)
{
    TecUtilLockStart(COMMANDPROCESSORID);
    if (TecUtilDataSetIsAvailable)
    {
        if (TecUtilFrameGetPlotType == PlotType_Cartesian2D)
        {
            TecUtilDialogGetVariables("Pick Variable to Sum",
                                     NULL,
                                     NULL,
                                     NULL,
                                     &Variable,
                                     NULL,
                                     NULL);
            BuildDialog1(MAINDIALOGID);
            TecGUIDialogLaunch(Dialog1Manager);
            TecUtilProbeInstallCallback(MyProbeCallback,
                                      "Summing Probed Values");
        }
        else
            TecUtilDialogErrMsg("Plot type must be 2D cartesian.");
    }
    else
        TecUtilDialogErrMsg("Frame does not contain a dataset "
                             "with which to probe.");
    TecUtilLockFinish(COMMANDPROCESSORID);
}
```

This example is limited to 2D plots.

Step 3 The MyProbeCallback function

The `TecUtilProbeInstallCallback(MyProbeCallback, "Summing Probed Values")` function calls the function `MyProbeCallback` each time a point is probed.

In `main.c`, note the function `MyProbeCallback` above `MenuCallback`:

```
static void STDCALL MyProbeCallback(Boolean_t IsNearestPoint)
{
    TecUtilLockStart(AddOnID);

    if (IsNearestPoint)
    {
        double ProbeValue = TecUtilProbeFieldGetValue(Variable);
        char Msg[100];
```

```

    Total = Total + ProbeValue;
    sprintf(Msg, "The total is: %f", Total);
    CHECK(strlen(Msg) < sizeof(Msg));

    TecGUILabelSetText(TheTotalis00_LBL_D1, Msg);
}
else
    TecUtilDialogErrMsg("You must hold down the Ctrl key when probing");

    TecUtilLockFinish(AddOnID);
}

```

Each time a point is probed, the callback checks to see if it was probed while holding down Ctrl. If it was, it gets the value of the variable, adds it to the running total, and changes the text displayed on the dialog to reflect this.

SumProbe is complete. Recompile and load it into Tecplot 360.

Step 4 Exercises

1. Enhance *SumProbe* to allow for interpolated values while probing.
2. Add a Clear button to the dialog to zero out the summed values.

Part 7 Common Add-ons

Building Data Set Reader Add-ons

A data set reader add-on allows you to load non-Tecplot format data into Tecplot 360. Once registered with Tecplot 360, the data reader can then be accessed with the **Load Data File(s)** dialog and referenced with the `!ReadDataSet` macro command. This then enables layout files generated by Tecplot 360 to reference your data set reader add-on.

Data set readers are divided into two different types: “data set converters” and “data set loaders.” When either of these is created using an add-on script as described in [Chapter 2: “Creating Add-ons on Linux/Macintosh Platforms”](#), the script will create the skeleton code you need.

26 - 1 Converters Versus Loaders

Data can be imported into Tecplot 360 using *loader* or *converter* add-ons. A *loader* must display a dialog for the user to enter the parameters needed to load the data; file name, skip values, and so forth. A *converter* is used when simple proprietary data files need to be read into Tecplot 360 and it is not necessary to use complex options to decide which portions of the data should be loaded. Converters are not as efficient as loaders because the creation of an intermediate file is a part of the operation.

26 - 1.1 How a Data Converter Works

A data *converter* is a special type of add-on which can read data in a custom file format and import it into Tecplot 360. It does this by reading the data and writing out a temporary binary data file. Tecplot 360 loads this temporary file and then discards it. Tecplot 360 queries the user for a file name, then passes it to the converter. If you need to query users for information other than file names, you must use a data *loader*. (Data loaders are discussed in [Chapter 27: “Creating a Data Loader”](#).) Given the file name, the procedure used by a converter to import that data is similar to creating a Tecplot 360 binary file using the `TecIO` functions. (See [“Binary Data” in the Data Format Guide](#) for more information on using the `TecIO` library.) Refer to [Chapter 29: “Creating a Data Converter”](#) for additional information, as well as a tutorial example.

26 - 1.2 How a Data Loader Works

A data loader is a special type of add-on which can load data of different file formats into Tecplot 360. Your Tecplot 360 installation includes loaders for several popular file formats, including FLUENT® and CGNS. Data loaders appear under “Load Data File(s)” in the **File** menu. Data loaders sometimes have custom dialogs for collecting loading parameters.

An add-on informs Tecplot 360 that it is a data loader by:

- Registering as a data loader by calling the `TecUtilImportAddLoader` function. This is called from the `InitTecAddOn` function in *main.c*.
- Exporting an interface callback function, named `LoaderSelectedCallback`, called by Tecplot 360 when you select the Load Data File(s) option from the **File** menu. This usually displays a dialog to collect loading parameters. After collecting the parameters the add-on will call the loader function to load the data.
- Exporting a loading callback function, `LoaderCallback`, which is called by Tecplot 360 to load the data. This is used whenever a file is loaded by macro or by selecting “Load Data File(s)” from the **File** menu.

After it's been registered, the *loader* add-on waits for:

- The user to select it from the **Load Data File(s)** option. Then Tecplot 360 calls the registered interface callback.

- or -

- Tecplot 360 to process the `$!READDATASET` macro command. Then Tecplot 360 calls the loader callback. (In this case, the add-on will not display a dialog.)

Refer to [Chapter 27: “Creating a Data Loader”](#) for additional information, as well as a tutorial example.

Creating a Data Loader

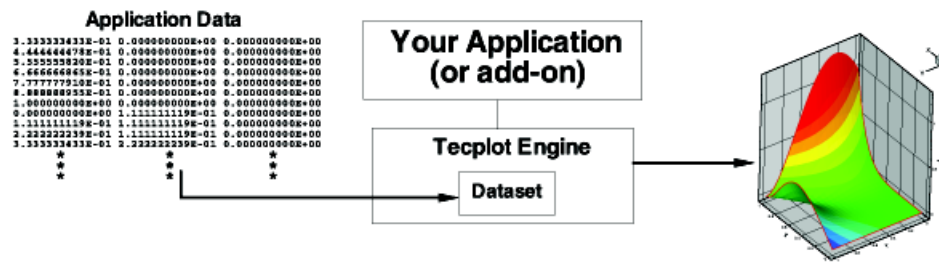


Figure 27-1. The Goal - Produce a plot from your data.

Before creating a plot with , you must first connect your data. Your data may be stored as one or more files on the hard disk, or local storage in your application. It may even be calculated from another data source via a formula. Once your data is connected, the primary steps for loading your data are independent of your data source.

Code used to load data can be found in a Tecplot add-on (which is often referred to as a "data loader add-on"). This chapter discusses the overall approach to loading data. We recommend that you isolate the actual data loading functionality in its own module. The remainder of this chapter refers to this module as the "core loader function".

27 - 1 Data Hierarchy

In the Tecplot Engine, your data is divided into subsections called “zones”. Zones are simply a way of partitioning a large amount of data into smaller pieces and are often used to represent a given solution at specific times and/or solutions clustered in different spatial locations (or both).

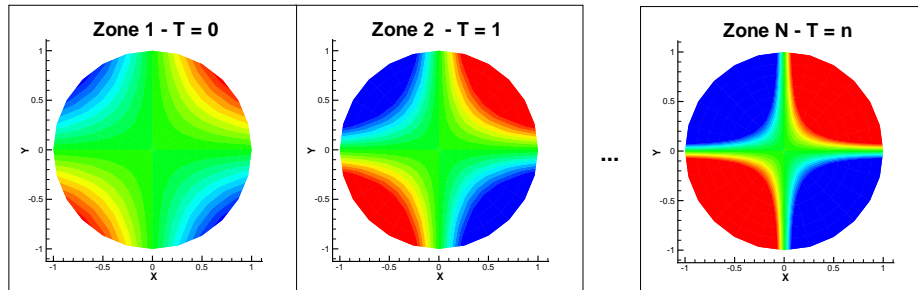


Figure 27-2. A collection of zones - defined by solution time.

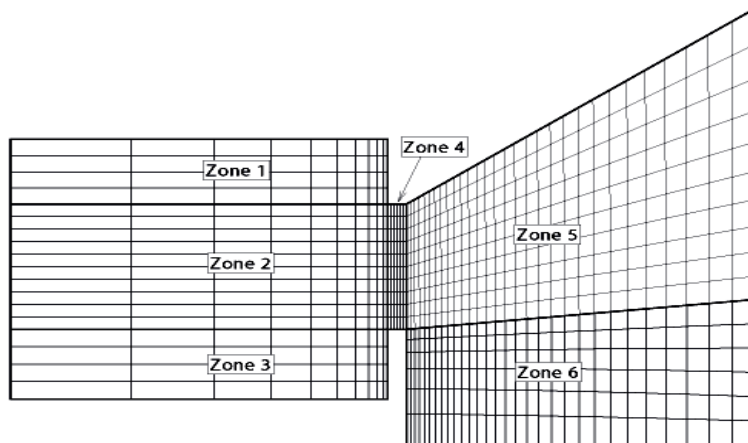
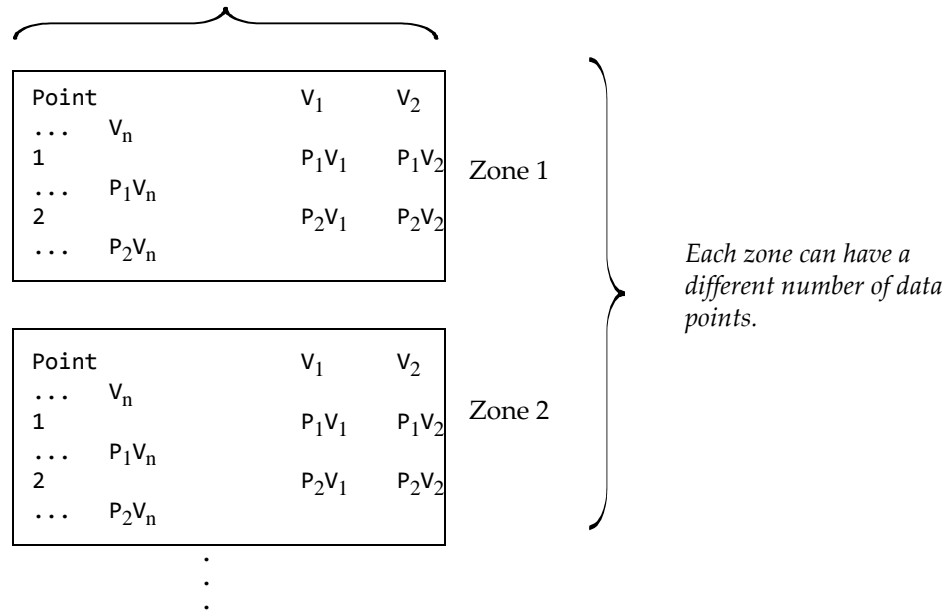


Figure 27-3. A collection of zones - defined by spatial

A Tecplot Zone is composed of a set of variables (n) with each variable holding a set of data point values (p). Each zone may have any number of data points. A value must be supplied for every data point in every variable.

*All zones in a given dataset
must have the same number
of variables.*



Cells are formed by connecting the data points. The method used to form cells identifies the zone "Type".

Zone Type ^a	How Cells Are Determined
Ordered	Points are logically connected based on the node ordering. An I-ordered zone is a logical 1D array of points. IJ-ordered Zones are a 2D array of points and IJK-ordered 3D. J, K, JK, IK-ordered Zones are also possible.
FE Line Segment	Pairs of points are connected to each other. Not all of the points have to be connected and not all connections have to be contiguous (i.e. there can be many clusters of cells).
FE-Triangle	Surface Cells are formed by connecting three points.
FE-Quad	Surface Cells are formed by connecting four points.
FE-Tetra	Volume Cells are formed by connecting four points.
FE-Brick	Volume Cells are formed by connecting eight points.
Polygonal	Finite-element surface cells with an arbitrary number of points.
Polyhedral	Finite-element volume cells with an arbitrary number of faces.

a. For an in-depth discussion of Zone Types, refer to the [User's Manual](#).

Zones are grouped together to form datasets. A dataset manages your data in the Tecplot Engine.

27 - 2 Primary Data Load Steps

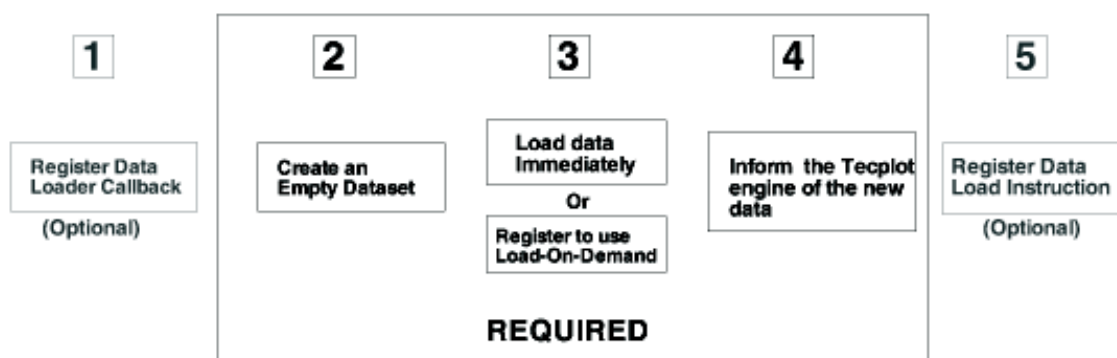


Figure 27-4. The six Data Load Steps. Steps 2, 3, and 4 are required and steps 1, 5 and 6 are optional.

[Figure 27-4](#) illustrates the primary steps for loading data into the Tecplot Engine. Data Load Steps 2 through 4 are required and steps 1, 5 and 6 are optional. Data Load Step 3 is the most complex step. It is where the data is loaded or attached, and also where you have the option to specify whether to use immediate loading or load-on-demand.

Detailed descriptions of each Data Load Step are provided in the remainder of this section. Detailed examples of each type of loading are provided in [Section 27 - 4 “Immediate Loading”](#) and [Section 27 - 5 “Load-on-demand”](#).

Step 1 Data Loader Registration (Optional)

Prior to loading any data, you may choose to register a data loader. If you choose to both register the data loader and register the data loading instructions ([Data Load Step 5: Data Journaling](#)), you will be able to save the entire state of a plot in a layout file without having to include the dataset (i.e. a layout package file) or write out the dataset.

To register your data loader call `TecUtilImportAddLoader`. (See complete examples in [Section “Immediate Loading Examples”](#) and [Section “Load-on-demand Examples”](#)). This call should occur during the initialization of your add-on or application. It should only be done once, even if the data loader is used multiple times in a given session.

Step 2 Creating the Dataset

Before you load the data, you must create the dataset. Creating the dataset is the first required step when building a data loader and is independent of your loading method (selected in [Data Load Step 3](#)). Data is created by calling `TecUtilDataSetCreate` to create an empty dataset. Calling `TecUtilDataSetCreate` also gives you the option to establish the number of variables per data point.

Step 3 Loading the Data

Once your data is created, you are ready to load the data. Before loading the data, you must decide whether to use [Immediate Loading](#) or [Load-on-demand](#). Immediate loading is often used for small or dynamic datasets, while Load-on-demand is used for large datasets.

Step 4 Inform the Tecplot Engine that Data Loading has Finished

Once you have created and loaded your data (i.e. Data Load Steps 2 and 3), you must inform the Tecplot Engine that data creation is finished. This is accomplished by sending a state change notification to the Tecplot Engine indicating that zones have been added. This is done by calling .

Step 5 Data Journaling (Optional)

After the required steps are complete (Data Load Steps 2-4) you have the opportunity to register data loading instructions with the Tecplot Engine. These instructions give the Tecplot Engine the ability to reproduce the dataset without saving a second copy using a layout file. A layout file orchestrates the data loading and the application of plot style. Refer to [Chapter 14: “Data Journaling”](#) for more information.

In order to activate journaling, call `TecUtilImportSetLoaderInstr`. Details of how to provide data journal instructions to the Tecplot Engine are described in [Section “Data Load Journal Instructions”](#).

Step 6 Setting an Initial Plot Style (Optional)

During the data load process a new dataset was created, setting the current frame to the Sketch plot type. If no other action is taken at this point, the Tecplot Engine will draw a blank sketch frame. To set the plot style simply call `TecUtilFrameSetPlotType`.



As a general rule we do NOT recommend that a data loader set the plot style. This exception might help if the data loader can determine the ideal plot type for the data.

27 - 3 Recommended Code Layering

The core coding for a data loader is the same for both Tecplot add-ons and Tecplot 360 parent applications. If your application will do only the minimal steps (i.e. Data Load Steps 2, 3 and 4 outlined in [Section “Primary Data Load Steps”](#)), then your job is straightforward.

Even in the simplest of cases, however, it is recommended that you layer your data loader to isolate the core loading functionality from other functionality.

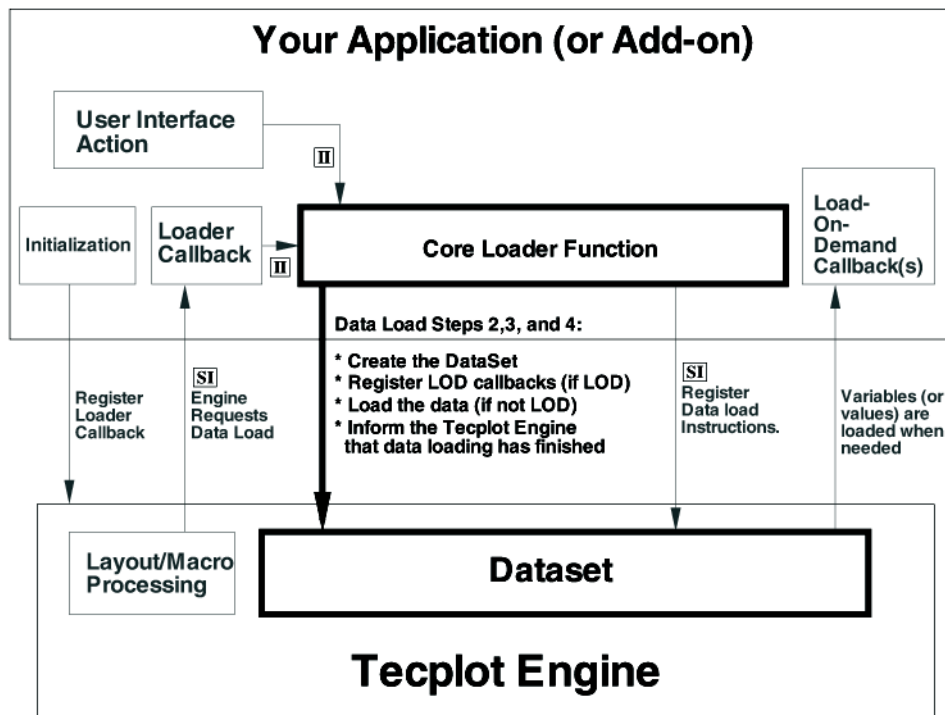


Figure 27-5. Interaction between the core components of a data loader and the Tecplot Engine. (LOD is the acronym for load-on-demand.)

The previous figure shows how the core components of a data loader interact with the Tecplot Engine. The bold items represent the required Data Load Steps. Items in gray represent components necessary to implement the optional Data Load Steps 1, 5 and 6.

27 - 3.1 Core Loader Function

We highly recommend that your core loader function receive instructions internally in some custom internal format ([II] notation in [Figure 27-5](#)). Alternatively, you may communicate externally with the Tecplot Engine. However, those communications must be in a stringlist format ([SI] notation in [Figure 27-5](#)).

All example loaders presented in the following sections take the approach of isolating the core loader function. The first loader presented ([Immediate Data Loading without Data Journaling](#)) implements only the portions of [Figure 27-5](#) that are bold. Each successive loader adds more functionality with the last two loaders ([Load Variable-on-Demand](#) and [Load Value-on-demand](#)) implementing all of the components in [Figure 27-5](#).

27 - 4 Immediate Loading

Immediate loading creates pre-allocated zones and copies your application data directly into variables within those zones.

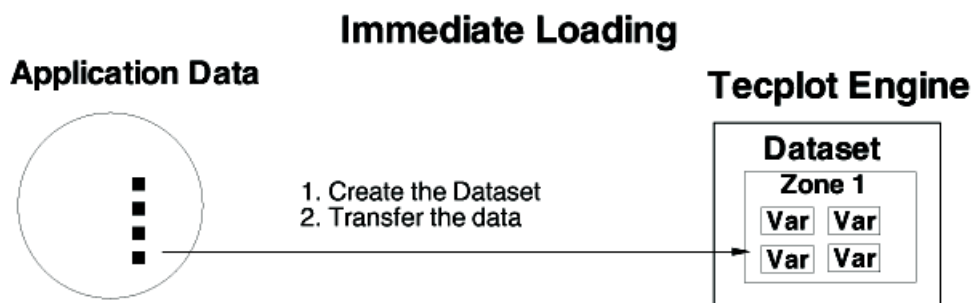


Figure 27-6. Immediate Data Loading.

Use immediate loading when:

- You are working with small to medium size data. A general rule of thumb is: if it will take less than two seconds to load your entire dataset using immediate loading, then immediate loading is sufficient.
- You prefer a simple solution for loading data.
- You are working with real-time data acquisition and you are always plotting the data. Refer to [Section 27 - 11 "Replacing Data"](#) for details on how to dynamically replace subsets of a dataset.
- You want to implement a quick solution for loading data now and, if needed, revisit it later and make it more efficient.

If none of the above criteria applies to your data, you will likely need to use load-on-demand. Refer to [Section 27 - 5 "Load-on-demand"](#) for examples of working with the different types of load-on-demand.

To use immediate loading, perform the following steps:

1. Create a zone with pre-allocated variables (i.e. Call `TecUtilDataSetAddZone` or `TecUtilDataSetAddZoneX` with `SV_DEFERVARCREATION` set to `FALSE`).
2. For each variable, obtain a writable reference.
3. Using the reference, transfer the data.

Pre-allocation of the variables is the key difference between immediate loading and load-on-demand.



Example 79: Immediate Data Loading without Data Journaling

This is the simplest method for loading data into the Tecplot Engine. Using this method will provide data for plotting. However, it will not allow you to save a layout that can reference the original dataset.



Your Tecplot product installation includes the complete source code for this example in the `adk/loadimmed` folder of your installation.

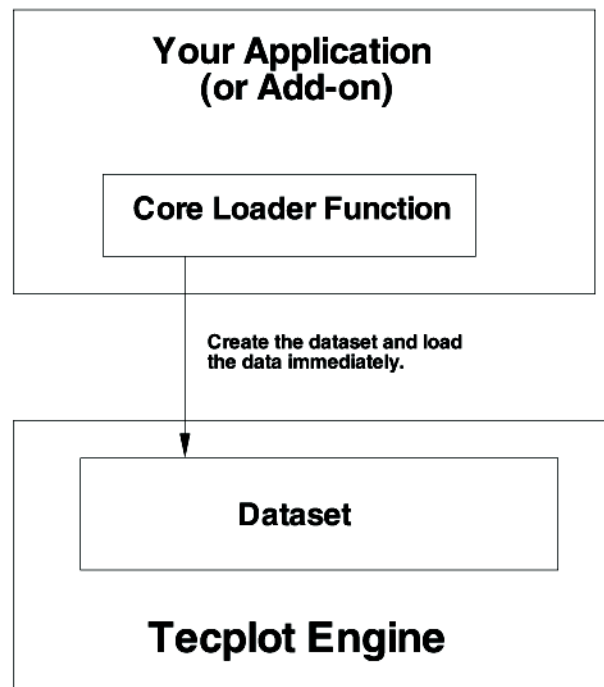


Figure 27-7. Immediate loading with no journaling.

Only the required Data Load Steps 2, 3 and 4 are necessary. Since journaling is not supported, the loader does not have to be registered.

The following code shows how to load data using the immediate loading method. The actual data values supplied to the dataset could come from any source, including: a data file, network sockets, or a formula. Here we will create a dataset containing a basic 10x10 zone where only X and Y are defined (determined by a simple formula). This code assumes that your application (or add-on) has already been initialized.

At this point, the data is loaded and a simple plot is made.



Example 80: Immediate Data Loading with Data Journaling

The immediate data loading method builds on the previous method and fills in the loader responsibilities to support data journaling (i.e. adds Data Load Steps 1 and 5). With data journaling a layout can make use of your custom data loader to re-read the data at a future time.



Your Tecplot product installation includes the complete source code for this example in the *adkloadjrn1* folder of your Tecplot product installation.

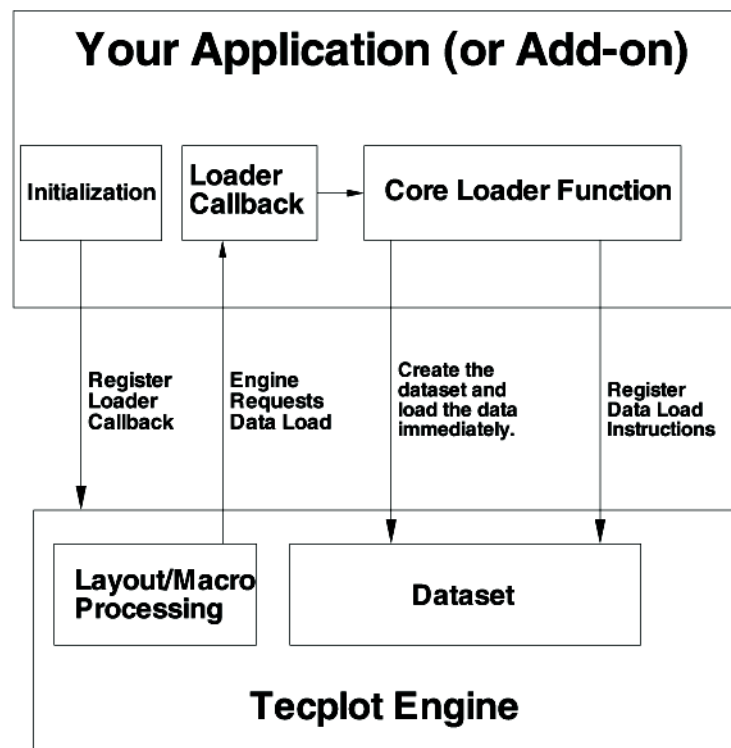


Figure 27-8. Immediate loading with journaling.

To support data journaling, we will need to do all six Data Load Steps. For Data Load Step 3, we will repeat what we did in the previous method, i.e. create a dataset containing a simple IJ-Ordered zone.

The following code should be placed in a separate module reserved for your data loader.

Core Loader Function

The Core Loader Function does the following:

1. Receives instructions in internal format.
2. Creates a dataset (i.e. Data Load Step 2).
3. Adds a zone, and assigns values to the variables (i.e. Data Load Step 3).
4. Informs the Tecplot Engine that data loading is finished (i.e. Data Load Step 4).
5. Registers load instructions with the engine (i.e. Data Load Step 5).
6. (Optional) Sets an initial plot style (i.e. Data Load Step 6).

This is the function called by the Tecplot Engine when data is to be loaded. It performs the following steps:

1. Convert the data load instructions from a stringlist to internal representation.
2. Call internal loader with the internal form of the instructions.

In this example, the code is slightly rearranged compared to the immediate loading method. The core data load coding is the same, with the exception of calling `TecUtilImportWriteLoaderInst` at the end to register the data load instructions with the Tecplot Engine.

To connect everything, we must register our loader with the Tecplot Engine using `TecUtilImportAddLoader`, which in turn uses `MyLoaderCallback` to isolate the parsing of load instructions from the core data load code.

27 - 5 Load-on-demand

Use Load-on-demand when:

- You are working with very large data.
- You plan to plot a subset of the data at a time.

[Figure 27-9](#) shows two common scenarios where using Load-on-demand is preferred.

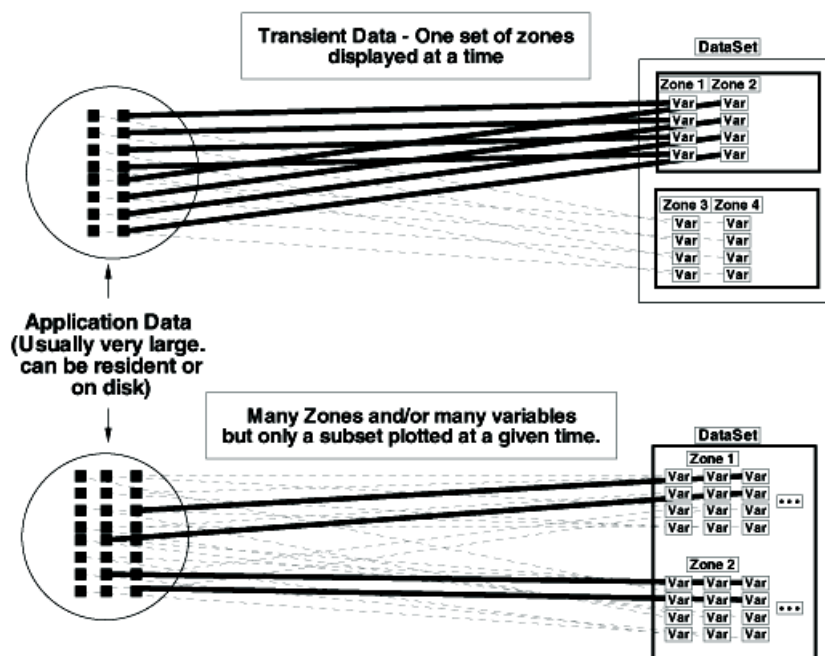


Figure 27-9. **Common Load-on-demand scenarios.**
The solid lines represent connections to variables that are currently loaded. The shaded lines represent variables that are registered for Load-on-demand but are not loaded.

Unlike [Immediate Loading](#), Load-on-demand does not force the Tecplot Engine to allocate space for variables as soon as zones are "defined". Instead each variable is initialized as a "passive" variable. Passive variables take up an insignificant amount of space and return a value of zero for all data locations until they are converted to active variables. Once a variable has been created (regardless of how it was created), it can be converted to use Load-on-demand. Load-on-demand then establishes a set of callbacks to manage loading.

To use Load-on-demand, perform the following steps:

1. Create zones using `TecUtilDataSetAddZoneX` (with `SV_DEFERVARCREATION` set to `TRUE`). At this point, all of the variables in the zone are passive.
2. Call either `TecUtilDataValueAutoLOD` or `TecUtilDataValueCustomLOD` to associate each variable with a set of Load-on-demand callbacks.
3. Write Load-on-demand callback functions as needed.

There are three types of Load-on-demand:

Type of Load-on-demand	Use when:
Auto-load Variable-on-demand	<p>Data resides on disk.</p> <p>Offsets to data values within files are easily determined.</p> <p>Data values are arranged in "Tecplot Order".^a</p> <p>There is no need to write special code to intervene in the loading process.</p>

Table 27 - 1: Types of Load-on-demand

Load Variable-on-demand	Data resides on disk or is locally resident. Data is not in "Tecplot Order" ^a and/or there is a need to intervene in the loading process.
Load Value-on-demand	Data is very large. Data values are generated by formula.

Table 27 - 1: Types of Load-on-demand

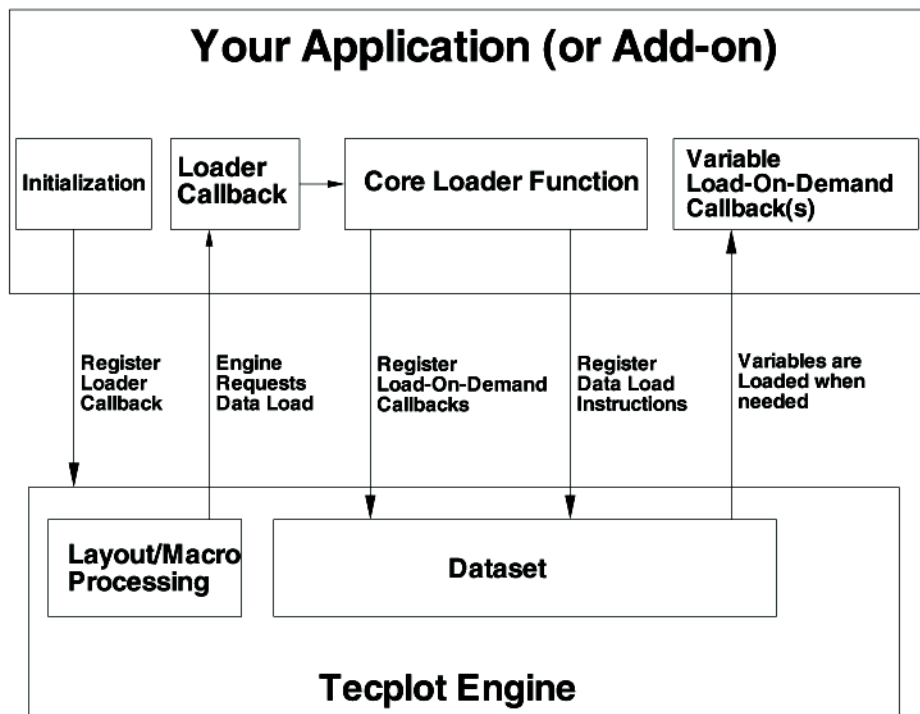
- a. For ordered data (i.e. IJK data) this is defined as a logical 1D, 2D, or 3D array (for each field variable) where I moves fastest, J moves second fastest, and K moves slowest. For Finite-element data, the field data is stored in simple 1D arrays.

An example for each type of load-on-demand is provided in the remainder of this section.

27 - 5.1 Auto Load-variable-on-demand

If your doesn't need to control when variables are loaded and unloaded, and if your cell-centered variable values (if any) are listed in a form accepted by the Tecplot Engine, you should use auto-load variable-on-demand.

Auto-load variable-on-demand is the simplest of the three load-on-demand methods. This is because the auto load callbacks are provided for you by the Tecplot Engine.

*Figure 27-10. Auto-load Variable-on-demand*

Auto-load variable-on-demand requires that you implement all six Data Load Steps from [Figure 27-5](#). In Data Load Step 3, we will register to use auto-load variable-on-demand.

With auto-load variable-on-demand, the Tecplot Engine controls all loading and unloading of variables. Your application or add-on only needs to supply the name of the data file and how to locate the data within the file. This is done with the `TecUtilDataValueAutoLOD` function.

The prototype for `TecUtilDataValueAutoLOD` is:

```
Boolean_t TecUtilDataValueAutoLOD(
```

```

        EntIndex_tZone,
        EntIndex_tVar,
        DataValueStructure_eDataValueStructure,
        const char*FileName,
        FileOffset_tOffset,
        EntIndex_tStride,
        Boolean_tIsDataNativeByteOrder
    )

```

Parameter descriptions are:

DataValueStructure	How cell-centered data in your files is formatted.
Offset	The position in the file of the first value for the variable.
Stride	The distance between values for the variable in the file: 1 for Block data; Number of variables for Point data ^a .
IsDataNativeByteOrder	Indicates if the byte ordering of the data in the file matches the machine's native byte ordering. One common way to determine this up front is to place a known value (like "1") in the first four bytes of a file, and then read that value with one byte ordering. If it does not return "1", then try another byte ordering.

a. In order to use point format, the data must be nodal.

Once a variable is registered using `TecUtilDataValueAutoLOD`, all loading, unloading and managing of the variable if altered is handled by the Tecplot Engine. For Block format files, if memory mapping is enabled, the variable is auto memory mapped.



Example 81: Using Auto Load-variable-on-demand

The following sample code assumes that the byte order is "Native". For this exercise, assume the Data Value Structure is ClassicPlus.



The code below shows the main components but is not complete. Your Tecplot product installation includes the complete source code in the `adk/loadauto` folder of its installation.

To support data journaling, we will need to do all six Data Load Steps. Unlike the immediate load methods, Data Load Step 3 will first add a zone (with `DeferVarCreation` set to `TRUE`) and then register auto load information with the Tecplot Engine.

The following code must be placed in your application or add-on where initialization is performed.

```

TecUtilImportAddLoader(LoaderCallback,
                      LoaderName.c_str(),
                      LoaderSelectedCallback,
                      NULL);

```

This code should be placed in a separate module reserved for your data loader. Include members as necessary. At a minimum you should have a member for a filename or list of filenames.

Core Loader Function

The following code is the Core Loader function and performs the following steps:

1. Receives instructions in internal format.
2. Pre-scans input files to collect the number of zones, zone sizes, etc.
3. Creates a dataset (i.e. Data Load Step 2).
4. Adds zones specifying deferred loading.
5. Registers auto load information for each variable in each zone.
6. Informs the Tecplot Engine that data loading is finished (i.e. Data Load Step 4).
7. Registers load instructions with the engine (i.e. Data Load Step 5).
8. Sets an initial plot style (i.e. Data Load Step 6). (Optional)

Using internal instructions, open data files and collect information on number of zones, number of variables, zone sizes, etc. At a minimum, VarNames must be populated here.

For each zone to be loaded, add the zone to the Tecplot Engine with deferred variable loading set to TRUE.

Other calls set dimensions, zone name, etc. This information is either hard coded or obtained by opening the data files and reading header information.

Inform the Tecplot Engine that adding zones is finished. You will need to create a ZoneSet set and populate with all zone numbers being loaded.

This is the function called by the Tecplot Engine when data is to be loaded. It performs the following steps:

1. Convert the data load instructions from a stringlist to internal representation.
2. Call internal loader with the internal form of the instructions.

Code here parses instructions, checks for errors, and converts them to an internal representation (for example, update values for all members of a structure).

The Tecplot Engine will do the rest. When the engine needs a variable and that variable is not currently loaded into memory it will open the file, seek to the offset where the variable resides and load or memory map the variable.

27 - 5.2 Load Variable-on-Demand

Load Variable-on-demand is ideal for large data files if your application or add-on needs to be informed, or to control when the Tecplot Engine loads and unloads variables. Load Variable-on-demand allows your application or add-on to control the loading and unloading of variables with TecUtilDataValueCustomLOD.

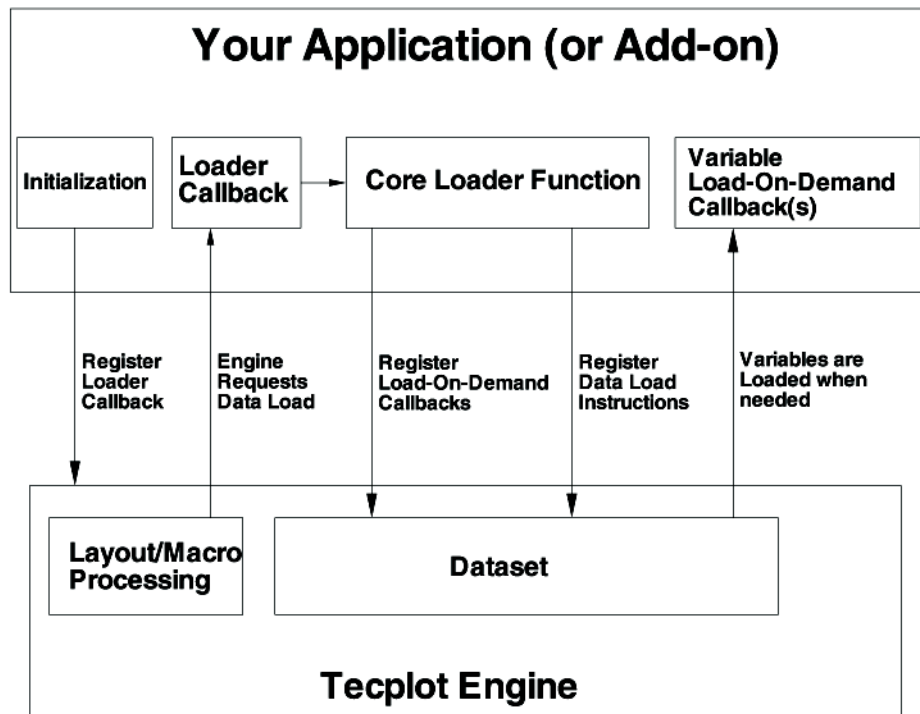


Figure 27-11. Load Variable-on-demand

Load variable-on-demand requires that you implement all six Data Load Steps from [Figure 27-5](#). In Data Load Step 3, we will register to use load variable-on-demand. Load variable-on-demand is orchestrated by calling TecUtilDataValueCustomLOD for each variable in each zone of your dataset.

Initialization and loader callback registration itself are the same for load variable-on-demand and load value-on-demand as they are for auto-load variable-on-demand

The prototype for TecUtilDataValueCustomLOD is:

```

Boolean_t TecUtilDataValueCustomLOD (
    EntIndex_t      Zone,
    EntIndex_tVar,
    LoadOnDemandVarLoad_pf VariableLoad,
    LoadOnDemandVarUnload_pfVariableUnload,
    LoadOnDemandVarCleanup_pfVariableCleanup,
    FieldValueGetFunction_pfGetValueFunction,
    FieldValueSetFunction_pfSetValueFunction,
    ArbParam_t ClientData)

```

Parameter descriptions include the following:

VariableLoad	Your function that reads the values for a variable and loads them into the Tecplot Engine.
VariableUnload	Your function that unloads a variable. Returning FALSE for this function will prevent unloading of the variable. Set this to NULL, in order to have the Tecplot Engine take responsibility for unloading. Refer to Section 27 - 5.3 "Load Value-on-demand" for additional information.

VariableCleanup	Your function that cleans up a variable.
GetValueFunction	Set to NULL for this load method.
SetValueFunction	Set to NULL for this load method.
ClientData	A pointer to an allocated item, containing all the information needed to load, unload or cleanup the variable when requested by the Tecplot Engine. The structure should include the file name, the offset to the first value for the current variable, and the number of points. It should also include settings such as the data type and the byte order. If data is in point format, the structure must also include the offset between values for a variable or the stride.

The `ClientData` and a field data pointer are stored for each variable, and passed by the Tecplot Engine to the appropriate function (Load, Unload, Cleanup) each time the Tecplot Engine needs to load or unload a variable. In the simple and most common case that a staging area does not need to be allocated, the Load/Unload/Cleanup functions are optional and only serve to provide notification of the Tecplot Engine's intent.

The process is as follows:

1. A request is made to load a dataset.
2. `LoaderCallback` is called with loading instructions.
3. Call `TecUtilDataSetCreate`.
4. For each zone, call `TecUtilDataSetAddZoneX` with `SV_DEFERVARCREATION=True`.
5. For each variable, call `TecUtilDataValueCustomLOD` with Load/Unload/Cleanup functions and `ClientData`.



Example 82: Using Load Variable-on-demand



This section concentrates on the core loader function. Source code for a complete sample using load variable-on-demand can be found in `adk/loadvar` below the distribution root directory.

For load variable-on-demand, you must place necessary information into the structure `ClientDataValues`. The structure provides information about the original file and how to load the data requested by the Tecplot Engine. This structure must be defined in a header file:

Registering Load Variable-on-demand Functions

The three functions used with `TecUtilDataValueCustomLOD` must be defined. The information that was registered with the command `TecUtilDataValueCustomLOD` is passed by the Tecplot Engine as part of a field data pointer to each function, each time a variable is loaded or unloaded.

We recommend that you add error checking each time you send and receive parameters for a function. To further improve your coding, perform the following steps:

- Check byte order.
- Add a parameter to `ClientData` that would allow the loader to skip over values to load data in Point format.
- Allow other data types besides double.

27 - 5.3 Load Value-on-demand

A single value of a variable will be accessed and loaded based on the index, time or other parameter set by the . Load value-on-demand uses the same command as load variable-on-demand - `TecUtilDataValueCustomLOD`.

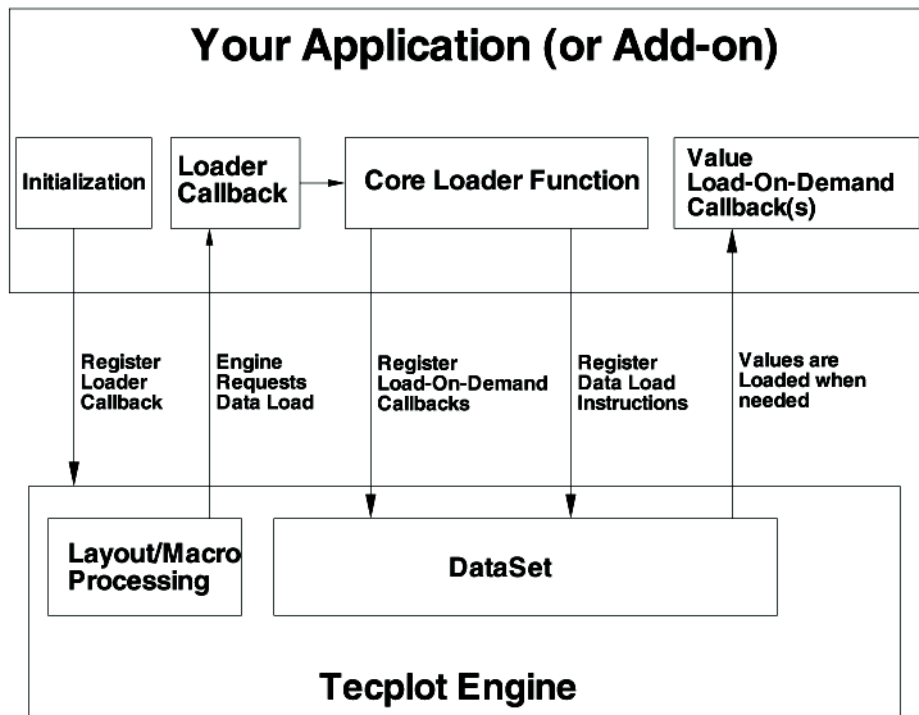


Figure 27-12. Load Value-on-demand

Load value-on-demand requires that you implement all six Data Load Steps from [Figure 27-5](#). In Data Load Step 3, we will register to use load-value-on-demand. Load value-on-demand is orchestrated by calling `TecUtilDataValueCustomLOD` for each variable in each zone of your dataset. Unlike load variable-on-demand, you need to register valid callbacks to set and get individual values.

Initialization and loader callback registration itself are the same for load value-on-demand and load variable-on-demand, as they are for auto-load variable-on-demand.



This section will concentrate on the core loader function. Source code for a complete sample using load value-on-demand can be found in `adk/loadvalue` below the distribution root directory.

The prototype for `TecUtilDataValueCustomLOD` for load value-on-demand is:

```
Boolean_t TecUtilDataValueCustomLOD
(
    EntIndex_t          Zone,
```

```

EntIndex_t
LoadOnDemandVarLoad_pf
LoadOnDemandVarUnload_pf
LoadOnDemandVarCleanup_pf
FieldValueGetFunction_pf
FieldValueSetFunction_pf
ArbParam_t
)
Var,
VariableLoad,
VariableUnload,
VariableCleanup,
GetValueFunction,
SetValueFunction,
ClientData

```

Parameter descriptions are:

VariableLoad	Your function that allocates the space for a staging area for data, reads the file and stores the values into the staging area. It does not place these values into the dataset.
VariableUnload	Your function that frees the space allocated for the staging area. Returning FALSE for this function will prevent unloading of the variable. Most s should set this to NULL, causing the Tecplot Engine to take responsibility for unloading the variable.
VariableCleanup	Your functions that frees the space allocated for the staging area and the ClientData.
GetValueFunction	Called when the Tecplot Engine needs to retrieve a value from the staging area. It will retrieve the appropriate value from the staging area for that variable and return it to the Tecplot Engine. You may register a unique GetValueFunction for different variables if desired.
SetValueFunction	Submit a value to field data. It is responsible for storing any changes to field data until the variable is cleaned up. Most s will set this to NULL, allowing the Tecplot Engine to take responsibility for storing changes.
ClientData	A pointer to an allocated structure defined by the writer, containing all the information needed to load, unload or cleanup the variable when requested by the Tecplot Engine.

The ClientData and a field data pointer are stored for each variable, and passed by the Tecplot Engine to the appropriate function (Load, Unload, Cleanup, GetValue or SetValue) each time the Tecplot Engine needs to load or unload a variable. Both the VariableLoad and VariableUnload functions need to lock the Tecplot Engine at their beginning and unlock before they return.

The process is as follows:

1. A request is made to load a dataset.
2. LoaderCallback is called with loading instructions.
3. Call TecUtilDataSetCreate.
4. For each zone, call TecUtilDataSetAddZoneX with SV_DEFERVARCREATION=True.
5. For each variable, call TecUtilDataValueCustomLOD with Load/Unload/Cleanup functions, one or more Get and Set functions and ClientData.

27 - 5.4 Load-on-demand Callbacks and Threads

The load-on-demand callbacks used in both load variable-on-demand and load value-on-demand must be thread safe. Data loaders that use third party libraries that may not be thread safe in any Load-on-demand functions must resort to using mutex locking to prevent threads from conflicting with each other. This issue can be solved by calling TecUtilThreadMutexLock at the beginning of the function and TecUtilThreadMutexUnlock at the end.

27 - 6 Journaling Data Load Instructions

Most loaders display a user interface to prompt the user for what data to load and how to load it. Regardless of how this information is obtained, your loader must construct a stringlist that represents

these instructions and then pass the instructions to the Tecplot Engine after the data has been loaded by calling `TecUtilImportSetLoaderInstr`.

The loading instructions can be in any format you want. However, we recommend that you follow the standard syntax. Standard loader instructions are specified as tag/value string pairs in the instruction string list. That is, the "tag" and "value" strings must be consecutive strings in the string list. Except for the standard syntax identifier tag (which must be the first string in the string list), tag/value pairs may appear at any ordinal location in the string list and may be interspersed with custom instructions (which are ignored by the Tecplot Engine). Note that each custom instruction must be in the Tag/Value form. Newline delimited strings should not be used to pair multiple values with one name. Instead, use tag/value pairs with the same name for each value.

The standard syntax tags are summarized as follows:

Tag Name	Value	Required?	Notes
STANDARDSYNTAX	1.0	YES	To use loader standard instruction syntax, the loader must have the this tag/value as the first two strings in the string list.
DIRNAME_[directory_name_identifier]	[Directory Name]	NO	If the loader uses a directory name as part of its instruction string, then use this tag. Normally this tag is not used.
FILENAME_[file_name_identifier]	[File Name]	NO	If your loader uses a filename as part of its instructions, use this tag.
FILELIST_[file_list_identifier]	[N] [FileName1, FileName2,...,FileNameN]	NO	If your loader uses a list of file names, use this tag. ^a

a. FILELIST_ is the only exception to the STANDARDSYNTAX name/value pair rule by being in the form "FILELIST_identifier" "N#files" "file1" "file2" "... "fileN".

You may use any additional name/value pairs that you want. However, keep in mind that these instructions will appear in layout and macro files. As such, they should be easier for the end-user to interpret.



Example 83: Adding Journal Instructions for Data Loading

For example, if your loader only needs to know the name of a file and a skip. Your loader instructions could have the following three name/value pair commands:

Name	Example Value	Required?	Default Value
STANDARDSYNTAX	"1.0"	YES	N/A
FILENAME_TOLOAD	"MyFile.txt"	YES	N/A
SKIP	"3"	No	"1"

They would be added to the instructions string list by appending strings, as shown below:

```
StringList_pa Instructions;
Instructions = TecUtilStringListAlloc();
TecUtilStringListAppendString(Instructions, "STANDARDSYNTAX");
TecUtilStringListAppendString(Instructions, "1.0");
TecUtilStringListAppendString(Instructions, "FILENAME_TOLOAD");
TecUtilStringListAppendString(Instructions, "MyFile.txt");
TecUtilStringListAppendString(Instructions, "SKIP");
TecUtilStringListAppendString(Instructions, "3");
Result = LoaderCallback(Instructions);
TecUtilStringListDealloc(&Instructions);
```

The Tecplot Engine would save the following command in a layout or macro:

```
$!READDATSET ' "STANDARDSYNTAX" "1.0" "FILENAME_TOLOAD" "MyFile.txt"
"SKIP" "3"
DATASETREADER = "MyLoaderName"
```

One advantage of using standard syntax is that it allows the Tecplot Engine to save loader instructions with relative or absolute path names. If a user saves a layout and elects to use relative path names, the Tecplot Engine will scan through the loader instructions for all file and directory names identified via the above mechanism, and replace them with paths relative to the layout file path.

27 - 7 Loading the Connectivity List

The method used to load the connectivity depends upon the type of finite-element used. Cell-based or classic finite elements (where the elements are triangular, tetrahedral, brick, etc.) use a node array. Face-based finite elements (polyhedral or polygonal) use face maps.

27 - 7.1 For Classic Finite Elements

Classic finite-element zones need to supply a connectivity list (also known as a node map) to define the elements in the zone. Each element must specify what nodes it uses by referring to the number of the node, based on the order of the connectivity information delivered to the Tecplot Engine. The order that these nodes are specified is important to create elements with the correct shape. The number of nodes that each element specifies depends on its type, e.g. a triangle specifies three nodes, whereas a brick (or hexahedron) specifies eight nodes. The order that the elements are defined in the connectivity list is the order that is maintained by the Tecplot Engine (i.e. the first element defined in the connectivity list is element 1 in the Tecplot Engine zone).

To load the connectivity list for a classic or cell-based finite element zone into the Tecplot Engine, use the following procedure:

1. Define the ClientData and register LOD callbacks for the connectivity list (or node map) using TecUtilDataNodeAutoLOD or TecUtilDataNodeCustomLOD.



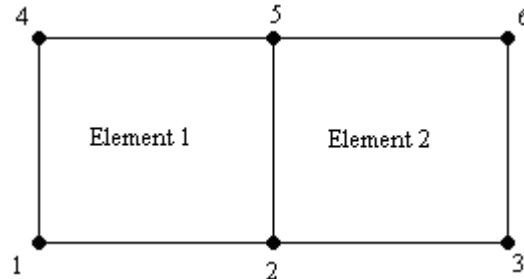
If you use TecUtilDataNodeAutoLOD, your data must be in binary block format.

2. In the LOD callbacks, retrieve the ClientData by calling TecUtilDataNodeGetClientData.
3. The reference is passed to the LOD callback by the Tecplot Engine.
4. Call TecUtilDataNodeArraySetByRef or TecUtilDataNodeSetByRef to pass the connectivity list.

5. Use the Unload callback from the LOD function to unload the facemap. This is handled by the Tecplot Engine if unload is set to NULL.
6. Use the cleanup callback from the LOD function to cleanup the client data.

Be sure to issue a StateChange_NodeMapsAltered state change after the LOD callbacks are registered.

For example, the configuration below has the following connectivity list:



```
1 2 5 4  #nodes for Element 1
2 3 6 5  #nodes for Element 2
```

27 - 7.2 For Face-based Finite Elements

Polyhedral and polygonal FE zones use face maps to store connectivity. Each face must specify all of the nodes that it uses. In addition, the left and right neighboring elements must be specified. Refer to [Section 10 - 2.1 "Polyhedral Finite-elements"](#) for background information.

If the data format your loader supports does not supply connectivity information, you can have the Tecplot Engine generate a face map from your element-to-node map. If connectivity information is available in the file, however, you should use it in preference to having it generated, even if some work is needed to get it into the format the Tecplot Engine requires. This usually performs better and also preserves the exact connectivity specified in the file.

If your data does not include connectivity information, it may be impossible to know how many unique faces are in the data until the data has been loaded. For this reason, when you create a zone using TecUtilDataSetAddZone or TecUtilDataSetAddZoneX, the KMax parameter (which indicates the number of faces for a face-based FE zone) may be unspecified or zero. The Tecplot Engine will fill in this value later when it generates the face map.

Use the following procedure to load or generate a face map.

1. Define the ClientData and register LOD callbacks for the facemap using TecUtilDataFaceMapCustomLOD.

```
/*
   Client data needed by facemap load-on-demand callback (in *.h)
   (This is an example; your loader may need different client data)
*/
typedef struct
{
    char          *FileName;
    LgIndex_t     ZoneUniqueID;
    LgIndex_t     VarUniqueID;
    LgIndex_t     NumFaces;
    LgIndex_t     NumBoundaryFaces;
    FileOffset_t  FaceDataOffset;
    /* etc. */
} FaceMapClientData_t;
```

```

/* Define client data and register LOD callbacks (in *.c) */
FaceMapClientData = (FaceMapClientData_t*)
    malloc(sizeof(FaceMapClientData_t))

if (FaceMapClientData != NULL)
{
    /* initialize client data */
}

TecUtilDataFaceMapCustomLOD(CurrentZone,
                             NumFaces, /* 0 to generate */
                             NumFaceNodes, /* 0 to generate */
                             NumFaceBndryFaces,
                             NumFaceBndryConns
                             LoadFaces,
                             NULL, /* Have Tecplot unload */
                             CleanupFaces,
                             (ArbParam_t)FaceMapClientData);

```

2. In the LOD callbacks, retrieve the ClientData by calling TecUtilDataFaceMapGetClientData.

```

/* Facemap LOD callback */
Boolean_t LoadFaces(FaceMap_pa FaceMap)
{
    FaceMapClientData_t *ClientData = NULL;

    ClientData = (FaceMapClientData_t *)
        TecUtilDataFaceMapGetClientData(FaceMap);

    /* now load the face map... */
}

```

3. The reference to the face map is generated by the Tecplot Engine and passed to the LOD callback. It is not necessary to call TecUtilDataFaceMapBeginAssign or TecUtilDataFaceMapEndAssign in a load on demand scenario.
4. Specify the face map using one of the two options detailed below, depending on whether the data file contains connectivity information.
5. Use the Unload callback from the LOD function to unload the face map. This is handled by the Tecplot Engine if Unload is set to NULL as it is in our example. If you provide an Unload callback, however, it must unload the facemap.
6. Use the cleanup callback from the LOD function to release any resources used by the face map loading process. Usually this will be just the client data. If the client data includes any Tecplot strings or other allocated objects, deallocate these first.

```

void STDCALL CleanUpFaceMap(FaceMap_pa FaceMap)
{
    FaceMapClientData_t *ClientData = NULL;
    ClientData = (FaceMapClientData_t *)
        TecUtilDataFaceMapGetClientData(FaceMap);

    TecUtilLockStart(AddOnID);

    if (ClientData->FileName)
        TecUtilStringDealloc(&ClientData->FileName);

    free(ClientData);
    ClientData = NULL;
}

```

When specifying the face map (step 4 above), choose one of these two options.

Option 1. If connectivity information is readily available, follow these steps to load it.

- a. Use `TecUtilDataFaceMapAssignNodes` to assign the face nodes to the face map. Specify how many nodes are in each face using the `NumFaceNodes` array and specify the the face nodes themselves in the `FaceNodes` array. You can use multiple calls to `TecUtilDataFaceMapAssignNodes` if this is more convenient.
- b. Define the right and left neighboring elements for each face using `TecUtilDataFaceMapAssignElems`. Again, you may use multiple calls to `TecUtilDataFaceMapAssignElems` for convenience, and may even intermingle these calls with calls to `TecUtilDataFaceMapAssignNodes`. The only requirement is that the information about the faces be delivered in the same order.
- c. If any of the boundary connections in your data are global or one-to-many, you will need to define boundary connections using `TecUtilDataFaceMapAssignBConns`. This includes faces that are partially obscured, have more than one neighboring element on a side, or have a neighboring element in a different zone.

```
TecUtilDataFaceMapAssignNodes(FaceMap,
                              ClientData->NumFaces,
                              NumFaceNodes,
                              FaceNodes);
TecUtilDataFaceMapAssignElems(FaceMap,
                              ClientData->NumFaces,
                              LeftElements,
                              RightElements);

if (ClientData->NumBoundaryFaces > 0)
    TecUtilDataFaceMapAssignBConns(FaceMap,
                                    ClientData->NumBoundaryFaces,
                                    BoundaryElemCounts,
                                    BoundaryElems,
                                    BoundaryZones);
```

Option 2. If connectivity information is not available in the data, simply call

`TecUtilDataFaceMapAssignElemToNodeMap` to have the Tecplot Engine generate the face map. You will need to provide the number of elements for the zone, arrays for the number of faces per element and nodes per face, and the actual nodes for each element, all of which you should already have from loading the nodes.

```
TecUtilLockStart(AddOnID);
TecUtilDataFaceMapAssignElemToNodeMap(FaceMap,
                                       TotalCells,
                                       FacesPerElem,
                                       NodesPerFace,
                                       ElemToNodeMap);
TecUtilLockFinish(AddOnID);
```

For polygonal zones, the `ElemToNodeMap` array should contain the index of all the nodes in each element. `NodesPerFace` should be `NULL`, since each polygonal face always has two nodes.

For polyhedral zones, the `ElemToNodeMap` array should contain the index of all the nodes for each face of each element. A brick-like polyhedron, for example, would be specified using twenty-four entries: four nodes for each of six faces (not just eight entries).

27 - 8 Loading Face Neighbor Data

Use face neighbors to specify connections between element faces of classic finite-element or ordered data to elements in other zones (global connections) or connections between classic finite-element or ordered elements (local connections) in the same zone.

The following method incorporates load-on-demand for face neighbor data:

1. Define the `ClientData` and register callbacks and define the `ClientData` with `TecUtilDataFaceNbrCustomLOD`.

2. The LOD callbacks retrieve the ClientData by calling TecUtilDataFaceNbrGetClientData.
3. The reference is passed to the LOD callback by the Tecplot Engine.
4. Load the face neighbor data using one of the two methods described in [Section 27 - 8.1 “Local one-to-one”](#) or [Section 27 - 8.2 “Global or one-to-many”](#) (depending on the face neighbor connection mode).
5. The Unload callback will unload the face neighbor data. This is handled by the Tecplot Engine, if set the NULL in the via TecUtilDataFaceNbrCustomLOD.

27 - 8.1 Local one-to-one

If all of the face neighbors in your dataset are perfect (i.e. local one-to-one), where each face has only one boundary and each boundary is local to the zone, you can specify the face neighbor connections by calling either TecUtilDataFaceNbrAssignByRef or TecUtilDataFaceNbrArrayAssignByRef.



If you opt to use both TecUtilDataFaceNbrAssignByRef and TecUtilDataFaceNbrArrayAssignByRef, please note that TecUtilDataFaceNbrArrayAssignByRef cannot be called after calling TecUtilDataFaceNbrAssignByRef.

27 - 8.2 Global or one-to-many

If any of the face neighbors in your dataset are global or one-to-many, you will need to:

1. Specify the face neighbor for each zone when you create the zone (using TecUtilAddZoneX).
2. You can later specify the face neighbor connections by calling TecUtilDataFaceNbrAssignByRef.



Face neighbors have expensive performance implications. Use face neighbors to manually specify only connections that are not implicitly defined via the connectivity list.

27 - 9 Transient Data

The examples provided in [Section 27 - 4 “Immediate Loading”](#) and [Section 27 - 5 “Load-on-demand”](#) are written using static data. Transient datasets are created in a similar manner to static datasets (i.e. using TecUtilDataSetAddZoneX). When adding transient zones, simply include the SV_STRANDID and SV_SOLUTIONTIME arguments in the ArgList for TecUtilDataSetAddZoneX. Refer to [Section 15 - 11 “Working with Transient Data”](#) for additional information regarding transient data, including strandID and solution time.

27 - 10 Appending Data with a Data Loader

A data loader can provide the capability to append data. Data loaders can also append data to an existing dataset. In order to append data, you must do the following:

1. Prior to appending data, query the existing dataset in the Tecplot Engine to make sure it is compatible with the data you wish to append.
2. [Optional] Prior to calling any TecUtil functions that modify data, you must suspend dataset marking in the Tecplot Engine by calling: TecUtilDataSetSuspendMarking(TRUE).
3. Do not call TecUtilDataSetCreate. Call TecUtilDataSetAddVar and/or TecUtilDataSetAddZone.
4. After appending data, call TecUtilStateChanged (or TecUtilStateChangedX) with StateChanged_ZonesAdded and/or StateChanged_VarsAltered - depending on what you did.
5. [Optional] Turn off the suspension of dataset marking: TecUtilDataSetSuspendMarking(FALSE);.

6. [Optional] As with normal loading, call `TecUtilImportSetLoaderInstr` to register the instructions needed for the append.

Performing steps two, five and six will cause the Tecplot Engine to journal this new command along with any other existing data creation/loading commands. These steps prevent the data journal in the Tecplot Engine from being invalidated with the activity of adding variables or zones. Refer to [Chapter 14: “Data Journaling”](#) for additional information.

27 - 11 Replacing Data

Once you have loaded data into the Tecplot Engine, you may wish to replace some or all of the data values to modify your plot. The method you use depends both on your data and your goals. Use the following table coupled with the [ADK Reference Manual](#) to replace data in the Tecplot Engine:

Goal	Function	Notes
Replace one or more variable values.	<code>TecUtilDataValueArraySetByRef</code> or <code>TecUtilDataValueSetByRef</code>	
Replace a zone while maintaining style settings.	<code>TecUtilDataSetAddZoneX</code>	When calling <code>TecUtilDataSetAddZoneX</code> to replace a zone, simply set the zone number for the new zone to that of the existing zone. The old zone will be overwritten and the new zone will be displayed according to the style settings of the previous zone.
Replace an entire dataset while maintaining style settings (Tecplot file format ONLY).	<code>TecUtilReadDataSet</code> (with the <code>ResetStyle</code> parameter set to <code>FALSE</code>)	If the data is in Tecplot file format, you may replace the data by calling this function. Refer to the Data Format Guide for instructions for creating Tecplot data format files.
Resize an existing zone.	<code>TecUtilZoneRealloc</code>	

27 - 12 Overriding Data Loader Instructions

When opening a layout, you are given the opportunity to override the data source instructions for the layout file. This allows you to apply a given layout to different data. If you choose this option, the Tecplot Engine will do the following:

1. Scan the layout file and determine the instructions needed to load the data for each referenced dataset.
2. Determine the data reader type required to load each dataset.
3. Allow the user to select the Reader/Instructions to override.

At this point the Tecplot Engine determines if there is a way to override the data source instructions. If your data loader has registered a data loader override function, then it will be called. If you opt to allow for data override, then create the function as follows:

```
Boolean_t MyLoaderInstructionOverride(StringList_pa sl)
{
/* First step is to decode the instructions and display the current
* settings. Use the TecUtilStringListxxx functions to read each
* instruction and to create a new set of instructions. Return TRUE if
* all goes well, otherwise FALSE. */
}
```

Adding this capability to your data loader is optional. To omit this capability supply NULL for the data loader override function parameter in the `TecUtilImportAddLoader` function.



If you only need your loader to allow override of the filenames, simply use the standard instruction syntax (see [Section "Data Load Journal Instructions"](#)).

Building Extended Curve Fit Add-ons

An extended curve fit add-on allows you to extend Tecplot 360's XY-plot curve fitting capability. Once registered with Tecplot 360, the extended curve fit can be activated from the Curves page of the **Mapping Style** dialog by: selecting the maps to which it will apply, selecting **Curve Type>Extended**, and choosing the curve fit from the list in the **Choose Extended Curve Fit** dialog. If the extended curve fit has settings which may be modified (optional), the dialog for modifying the settings may be accessed by selecting the [Curve Settings] button with the appropriate mapping(s) selected.

The extended curve fit consists of an initialization routine and one or more callback functions, which are called by Tecplot 360. These callback functions will call functions to compute the curve fit from the raw data. They may also call functions to launch dialogs for entry of user configurable settings and output of the curve fit coefficients.

Refer to [Example: Polynomial Integer Add-on](#) and [Example: Simple Average Add-on](#) for tutorials on creating curve-fit add-ons, as well as the sample add-on *crvravg*.

28 - 1 Registering the External Curve Fit

Extended curve fit add-ons are registered with Tecplot 360 by calling the following function:

```
Boolean_t TecUtilCurveRegisterExtCrvFit(
    const char
    GetXYDataPointsCallback_pf
    GetProbeValueCallback_pf
    GetCurveInfoStringCallback_pf
    GetCurveSettingsCallback_pf
    GetAbbreviatedSettingsStringCallback_pf
    AbbreviatedSettingsStringCallback);
    *CurveFitName,
    XYDataPointsCallback,
    ProbeValueCallback,
    CurveInfoStringCallback,
    CurveSettingsCallback,
```

which returns TRUE if the extended curve fit was added successfully.

- **CurveFitName** - the unique name given to the extended curve fit. This name is used in the list of extended curve fits in the **Choose Extended Curve Fit** dialog, launched from Extended option on the Curves page of the **Mapping Style** dialog.
- **XYDataPointsCallback** - the function that calculates the curve fit. This is the only function that required to create an extended curve fit add-on.

- **ProbeValueCallback** - the name of the function that will return the dependent value when the extended curve fit is probed at a given independent value. If *ProbeValueCallback* is set to NULL, Tecplot 360 will perform a linear interpolation based on the values returned by the *XYDataPointsCallback* function.
- **CurveInfoStringCallback** - the function that will create a string to be presented in the **Curve Information** dialog (accessed via the **Data** menu). *CurveInfoStringCallback* may be set to NULL if you do not wish to present a string to the **Curve Information** dialog.
- **CurveSettingsCallback** - the function that is called when the [Curve Settings] button on the Curves page of the **Mapping Style** dialog is selected (and the Curve Type is set to extended curve fit). *CurveSettingsCallback* may be set to NULL, if there are no configurable settings for the extended curve fit. If settings are changed, it is the responsibility of the add-on writer to inform Tecplot 360 of the change by calling the function *TecUtilCurveSetExtendedSettings*. This function is usually called when the [OK] button is selected on the add-on dialog.
- **AbbreviatedSettingsStringCallback** - the function that returns a short version of your curve settings string. This string will be presented in the Curve Settings text field on the Curves page of the **Mapping Style** dialog. *AbbreviatedSettingsStringCallback* may be set to NULL if you do not wish to assign anything to this string. Even if you do not assign anything to the *CurveSettings* string, you may define this function and return any string you wish. The Curve Settings text field on the Curves page of the **Mapping Style** dialog is roughly 50 characters wide, so strings longer than roughly 50 characters will be truncated.

Because extended curve fit add-ons aren't compatible with Tecplot 360 versions earlier than 10, it is important to check the Tecplot 360 version number before registering the curve fit. The version number may be obtained using the *TecUtilGetTecplotVersion* function. If this function returns a value less than 10, display an error message and don't call *TecUtilCurveRegisterExtCrvFit*.

If you are creating your add-on using the Tecplot GUI builder, and have created templates by running the *CreateNewAddOn* script (Linux/Macintosh):

- the add-on registration code (including the Tecplot 360 version test) will be found in the module *main.c*
- the callback functions listed previously will be found in the module *engine.c*
- **Curve Settings** dialog callback functions will be found in *guicb.c* (if configurable settings were requested).

Typically, you will not need to modify *main.c*.

28 - 2 Calculating the Curve Fit

Curve fitting is modeling the data with an analytical function containing adjustable parameters (curve fit coefficients). In many cases, the values of these coefficients are computed such that the curve is "best" in a statistical sense (the Least Squares method, for example). In other cases, the values of the coefficients are computed so that the curve passes through the raw data points and represents one possible interpolation between the points (splines, for example). Occasionally, a curve fit add-on may be used to compute and display a variable derived from the raw data points (for example, see the Running Average add-on provided as a sample with Tecplot 360).

Tecplot 360 displays the curve as a number of points connected by line segments. The number of points used is specified in the Curve Points field of the Curve page of the **Mapping Style** dialog. *XYDataPointsCallback* is the name of the callback function where the curve points of the curve fit are calculated. This is the only callback function that needs to be defined to create an extended curve fit add-on. It looks as follows:

```
Boolean_t STDCALL XYDataPointsCallback(
    FieldData_pa RawIndV,
```

```
FieldData_pa RawDepV,
CoordScale_eIndVCoordScale,
CoordScale_eDepVCoordScale,
LgIndex_t NumRawPts,
LgIndex_t NumCurvePts,
EntIndex_t XYMapNum,
char* CurveSettings,
double* IndCurveValues,
double* DepCurveValues);
```

Where:

- **XYDataPointsCallback** - the function that calculates the curve fit.
- **RawIndV** - the handle to the raw field data of the independent variable.
- **RawDepV** - the handle to the raw field data of the dependent variable.
- **IndVCoordScale** - an enumerated variable specifying whether the independent variable axis has a linear or log scale.
- **DepVCoordScal** - an enumerated variable specifying whether the dependent variable axis has a linear or log scale.
- **NumRawPts** - the number of raw field data values.
- **NumCurvePts** - the number of points that will construct the curve fit.
- **XYMapNum** - the map number that is currently being operated on.
- **CurveSettings** - the curve settings string for the current Line-map.
- **IndCurveValues** - a pre-allocated array of size *NumCurvePts* which the add-on will populate with the independent values for the curve fit.
- **DepCurveValues** - a pre-allocated array of size *NumCurvePts* which the add-on will populate with the dependent values for the curve fit.

The arrays, *IndCurveValues* and *DepCurveValues* are the main result of this function call.

Generally speaking, the *XYDataPointCallback* consists of two parts. The first part computes the curve fit coefficients, and the second populates the curve values arrays. The process for computing the curve fit coefficients is beyond the scope of this manual. For common techniques such as linear least squares and splines, there are several good books you can refer to for theory (see the [Numerical Recipes](#) series, published by the Cambridge University Press, for examples) and various libraries are available on the internet and elsewhere.

Regardless of what method you use to compute the curve fit parameters, you will need to extract data from the raw data arrays. This may be done with the *TecUtilDataValueGetByRef* utility. For example, with:

```
int I;
DepVar = TecUtilDataValueGetByRef(RawDepV, I)
```

DepVar will contain the *i*th element of the raw dependent variable array.

Populating the dependent variable arrays is relatively simple. Determine the spacing of the independent curve variable with the following code:

```
double IndVarMin, IndVarMax;
TecUtilDataValuesGetMinMaxByRef(RawIndV,
                                &IndVarMin,
                                &IndVarMax);
Delta = (IndVarMax-IndVarMin)/(NumCurvePts-1);
```

Then, set the independent and dependent curve variables in the following loop:

```
for (ii = 0; ii < NumCurvePts; ii++)
{
    IndCurveValues[ii] = ii*Delta + IndVarMin;
```

```

    DepCurveValues[ii] = CurveFunction(IndCurveValues[ii], Parameters);
}

```

If you use the `CreateNewAddOn` script (Linux/Macintosh), most of the code for populating the curve variable arrays is created for you. You will only need to modify the line that computes `DepCurveValues[ii]` (the default code sets this to the mean value of the raw dependent variable).

There are three items in the `XYDataPointCallback` parameter list that are not required for every case:

- The `CurveSettings` string is needed only if the curve fit has user configurable curve settings.
- `IndVCoordScale` and `DepVCoordScale` are needed only if the nature of the curve fit depends upon the axis scale used (log or linear). Generally this is not the case. Tecplot 360's standard curve fits, for example, are not dependent upon the axis scale.

28 - 3 Improving the Probe Value

Unless something special is done, a probe of an Line-map with an extended curve fit will perform a linear interpolation based upon the `IndCurveValues` and `DepCurveValues` arrays (discussed in previous section). This will be correct at the curve points and off by some error at points between the curve points. The magnitude of this error will depend upon the curve function and the data. The error will reduce as the number of curve points increases. If this error is acceptable, set `ProbeValueCallback` to NULL in the `TecUtilCurveRegisterExtCrvFit` call. If this error is unacceptable, provide a `ProbeValueCallback` function, which returns the dependent variable for a given independent variable. The syntax for this function is:

```

Boolean_t STDCALL ProbeValueCallback (
    FieldData_pa RawIndV,
    FieldData_pa RawDepV,
    CoordScale_eIndVCoordScale,
    CoordScale_eDepVCoordScale,
    LgIndex_t    NumRawPts,
    LgIndex_t    NumCurvePts,
    EntIndex_t   XYMapNum,
    char         *CurveSettings,
    doubleProbeIndValue,
    double*ProbeDepValue);

```

Where:

- **ProbeValueCallback** - name of your ProbeValueCallback function.
- **RawIndV** - the handle to the raw field data of the independent variable.
- **RawDepV** - the handle to the raw field data of the dependent variable.
- **IndVCoordScale** - an enumerated variable specifying whether the independent variable axis has a linear or log scale.
- **DepVCoordScale** - an enumerated variable specifying whether the dependent variable axis has a linear or log scale.
- **NumRawPts** - the number of raw field data values.
- **NumCurvePts** - the number of points that will construct the curve fit.
- **XYMapNum** - the map number that is currently being operated on.
- **CurveSettings** - the curve settings string for the current Line-map.
- **ProbeIndValue** - the value of the independent variable at the location of the probe.
- **ProbeDepValue** - the calculated value of the dependent variable at the location of the probe, based on the value of **ProbeIndValue**.

Much of the discussion in the previous section, applies here as well. The main difference is that you are computing a scalar variable, `ProbeDepValue`, instead of an array. It will be necessary to recompute the curve fit parameters and call the same curve function as discussed previously.

```
*ProbeDepValue = CurveFunction(ProbeIndValue, Parameters);
```

28 - 4 Providing Curve Fit Information

Once a user has implemented a curve fit, they will often want to view and/or save the Parameters computed by the curve fit for their data. This information is displayed in the **Curve Information** dialog, accessible from the **Data** menu. *CurveInfoStringCallback* is the name of the function that will create a string to be displayed in the **Curve Information** dialog (accessed via the **Data** menu). This callback may be set to NULL if you do not wish to present a string to the **Curve Information** dialog. The syntax for this function is:

```
Boolean_t STDCALL CurveInfoStringCallback (
    FieldData_pa      RawIndV,
    FieldData_pa      RawDepV,
    CoordScale_e      IndVCoordScale,
    CoordScale_e      DepVCoordScale,
    LgIndex_t         NumRawPts,
    EntIndex_t        XYMapNum,
    char              *CurveSettings,
    char              **CurveInfoString);
```

Where:

- **CurveInfoStringCallback** - the name of your *CurveInfoString* Callback function
- **RawIndV** - the handle to the raw field data of the independent variable.
- **RawDepV** - the handle to the raw field data of the dependent variable.
- **IndVCoordScale** - an enumerated variable specifying whether the independent variable ax- has a linear or log scale.
- **DepVCoordScale** - an enumerated variable specifying whether the dependent variable ax- has a linear or log scale.
- **NumRawPts** - the number of raw field data values.
- **XYMapNum** - the map number that is currently being operated on.
- **CurveSettings** - the curve settings string for the current Line-map.
- **CurveInfoString** - the string that is to be presented in the **Curve Information** dialog. The *CurveInfoString* must be allocated inside *CurveInfoStringCallback* using *TecUtilStringAlloc* as follows:

```
*CurveInfoString = TecUtilStringAlloc(Size, "CurveInfoString");
```

The string may then be written to using *sprintf*. In general, you should provide enough information in the *CurveInfoString* that the user can independently create the curve fit. Curve fit coefficients should be provided, for example, along with the ranges of applicability. You may also include statistical information about the curve fit or the data, as seen with the General curve fit included with your Tecplot 360 distribution.

28 - 5 Curve Fit Settings

Extended curve fit add-ons may have user configurable settings. If so, the settings for each Line-map are saved in a character string maintained by Tecplot 360. Your add-on must initialize this string, update it when settings are changed, and inform Tecplot 360 by calling *TecUtilCurveSetExtendedSettings* or *TecUtilXYMapSetCurve* (with the first parameter being "EXTENDEDSETTINGS"). Your add-on must also parse this string to extract the configurable curve fit parameters.

The curve settings string can be in any format you like. However, keep in mind that these instructions will appear in layout and macro files, so they should be somewhat readable. Also, the string must not contain single quotes since the entire curve settings string is surrounded by single quotes in Tecplot 360 layout and macro files.

CurveSettingsCallback is the name of the function that is called when the [Curve Settings] button on the Curves page of the **Mapping Style** dialog is selected while the extended curve fit is set as the Curve Type. If *CurveSettingsCallback* is set to NULL in the call to *TecUtilCurveRegisterExtCrvFit*, there are no configurable settings for the extended curve fit. The syntax for this function is:

```
void STDCALL CurveSettingsCallback(Set_pa LineMapSet, StringList_pa SelectedXYMapSettings);
```

Where:

- **CurveSettingsCallback** - the name of your function that launches your extended curve settings dialog. *CurveSettingsCallback* usually sets any global variables and launches the curve settings dialog.
- **LineMapSet** - the set of Line-maps that are selected in the **Mapping Style** dialog.
- **SelectedXYMapSettings** - a string list of the curve settings for the Line-maps that are selected on the **Mapping Style** dialog.

You must provide one or more dialogs to prompt the user for the curve fit settings. We strongly recommend that this be a modal dialog to minimize the complexity of monitoring for changes while the dialog is up. When the [OK] button is selected in the **Curve Settings** dialog, update the curve settings string and inform Tecplot 360 by calling *TecUtilCurveSetExtendedSettings* or *TecUtilXYMapSetCurve* (with the first parameter being "EXTENDEDSETTINGS"). Remember that the settings must be modified for all Line-maps that were selected when the dialog was launched. These map numbers are provided by Tecplot 360 in *LineMapSet*. If *TecUtilCurveSetExtendedSettings* is used, you must loop through the maps in *LineMapSet* and set each one. In contrast, *TecUtilXYMapCurve* only needs to be called once, with *LineMapSet* as an argument.

If you are creating your add-on using Tecplot GUI Builder, and have created templates by running the *CreateNewAddOn* script (Linux/Macintosh) (with configurable settings requested), the *CurveSettingsCallback* function will be found in the module *engine.c*. This function, which saves *XYMapSet* and *XYMapSettings* in global variables and launches the **Curve Settings** dialog, will probably not need to be modified. The curve settings dialog callback functions will be found in *guicb.c*.

28 - 6 Creating the Curve Settings Text Field

Below the Curve Settings button in the Curve page of the **Mapping Style** dialog is a text field providing a brief description of the curve settings for each map. The contents of the text field are set in the *AbbreviatedSettingsStringCallback* function. If you do not want to provide an *AbbreviatedSettings* string, set *AbbreviatedSettingsStringCallback* to NULL in the call to *TecUtilCurveRegisterExtCrvFit*.

Even if you do not have configurable settings, you may define this function and return any string you wish. The Curve Settings option on the Curves page of the **Mapping Style** dialog is roughly 50 characters wide, so strings longer than roughly 50 characters will be truncated. The syntax for this function is:

```
void STDCALL GetAbbreviatedSettingsStringCallback(
    EntIndex_t      XYMapNum,
    const char*      CurveSettings,
    char**           AbbreviatedSettings);
```

Where:

- **AbbreviatedSettingsStringCallback** - the name of your function that will return the Abbreviated Settings string.
- **XYMapNum** - the map number that is currently being operated on.
- **CurveSettings** - the string that Tecplot 360 maintains which contains the extended curve fit settings for the current Line-map.

- **AbbreviatedSettings** - the short form of the *CurveSettings* that are passed into your function by Tecplot 360. The *AbbreviatedSettings* string must be allocated inside the *AbbreviatedSettingsStringCallback* function using *TecUtilStringAlloc* as follows:

```
*AbbreviatedSettings = TecUtilStringAlloc(Size, "AbbreviatedSetting");
```

The string may then be written to using *sprintf*.



Example 84: Polynomial Integer Add-on

The Tecplot 360 add-on you will build in this tutorial is an example of an extended curve-fit add-on that does not have any settings which may be configured. This add-on, called *PolyInt*, will add an option to the single selection list that is launched by the Curve Type>Extended option on the Curves page of the **Mapping Style** dialog. The code in this example is platform-independent.

This add-on will perform three operations:

- Calculate the curve-fit of discrete XY-data. This is the only option that is required.
- Supply Tecplot 360 with a dependent value when the plot is probed
- Present a string to the **Curve Information** dialog

All of the example of source code shown in this manual is included in the Tecplot 360 distribution and are found in the *adk/samples/polyint* subdirectory below the Tecplot 360 home directory.



Before proceeding with this tutorial, please read [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) and/or [Chapter 3: "Creating Add-ons on Windows Platforms"](#).

Step 1 Getting Started

PolyInt uses the source code files created by the *CreateNewAddOn* script (Linux/Macintosh). The project name and add-on name will both be *PolyInt*.

When running *CreateNewAddOn*, answer the questions as follows:

- **Project Name (Base name)** - PolyInt
- **Add-on name** - PolyInt
- **Company Name** - [Your company name]
- **Type of add-on** - Extended Curve-Fit
- **Language** - C
- **Allow Configurable Settings** - No
- **Create callback function for more accurate probing** - Yes

After running the *CreateNewAddOn* script, you should have the following files:

```
engine.c          main.c
ADDGLBL.h        ENGINE.h
```

You will also have other files specific to your platform, but the files listed are the only ones we will be modifying. The purpose of each file will be explained in detail as we proceed through the tutorial.

At this point, you should verify that you can compile your add-on and load it into Tecplot 360. If you are unable to compile or load your add-on, we recommend that you refer to [Chapter 2: “Creating Add-ons on Linux/Macintosh Platforms”](#) or [Chapter 3: “Creating Add-ons on Windows Platforms”](#) before proceeding.

1. *main.c* - This file contains the add-on registration routine. If you open the file, you will see a call to `TecUtilCurveRegisterExtCrvFit`. This function registers the extended curve-fit add-on with Tecplot 360. In *main.c*, the call to `TecUtilCurveRegisterExtCrvFit` will appear as follows:

Notice that parameters five and six are NULL. This is because this add-on has no settings

```
TecUtilCurveRegisterExtCrvFit(ADDON_NAME,
                             XYDataPointsCallback,
                             ProbeValueCallback,
                             CurveInfoStringCallback,
                             NULL, /* CurveSettingsCallback */
                             NULL); /*
AbbreviatedSettingsStringCallback */
```

which may be configured.

2. *ENGINE.h* - Open *ENGINE.h* and verify that the following lines exist:

Each of these functions are defined in *engine.c*.

```
extern Boolean_t STDCALL XYDataPointsCallback(
    FieldData_pa RawIndV,
    FieldData_pa RawDepV,
    CoordScale_e IndVCoordScale,
    CoordScale_e DepVCoordScale,
    LgIndex_t NumRawPts,
    LgIndex_t NumCurvePts,
    EntIndex_t XYMapNum,
    char *CurveSettings,
    double *IndCurveValues,
    double *DepCurveValues);
extern Boolean_t STDCALL CurveInfoStringCallback(
    FieldData_pa RawIndV,
    FieldData_pa RawDepV,
    CoordScale_e IndVCoordScale,
    CoordScale_e DepVCoordScale,
    LgIndex_t NumRawPts,
    EntIndex_t XYMapNum,
    char *CurveSettings,
    char **CurveInfoString);
extern Boolean_t STDCALL ProbeValueCallback(
    FieldData_pa RawIndV,
    FieldData_pa RawDepV,
    CoordScale_e IndVCoordScale,
    CoordScale_e DepVCoordScale,
    LgIndex_t NumRawPts,
    LgIndex_t NumCurvePts,
    EntIndex_t XYMapNum,
    char *CurveSettings,
    double ProbeIndValue,
    double *ProbeDepValue);
```

3. *engine.c* - When the source files are created, they include code to compute a simple average of the dependent values, by default. This code is not needed for this add-on and should be deleted. Delete the `SimpleAverage` function and all of the code in the callback functions (do not delete the function declarations themselves).

In *engine.c* we will define the three callbacks that are listed previously. First we will deal with the function that actually performs the curve-fit. The function is called `PolyInt`. It is based on a method given in the Stineman article from *Creative Computing* (July, 1980)¹. Much of this tutorial will focus on manipulating the data into a form that the `PolyInt` function can use. A description of the algorithm is beyond the scope of this tutorial.

The `PolyInt` function takes an array called `Data` and some information about the contents of the array. The `Data` array is separated into four separate blocks.

- **Block 1:** Raw independent data values.
- **Block 2:** Raw dependent data values.
- **Block 3:** Calculated independent values (based on the number of points on the calculated curve).
- **Block 4:** Calculated dependent values (to be filled in by `PolyInt` function).

We will also pass the indices of the start of each block, the number of raw data points, and the number of points on the calculated curve to the `PolyInt` function.

Note the following code in *engine.c* immediately following the last `#include` statement:

```
/**
 * Interpolate y=f(x) using the method given in Stineman article from
 * Creative Computing (July 1980). At least 3 points are required for
 * interpolation. If fewer than three points are supplied, then use
 * linear interpolation.
 *
 * Data is treated as a 1 based array, while lx,ly,lxn,lyn are treated
 * as 0
 * base.
 *
 * @param npts
 *   number of original data points
 * @param lx
 *   location of x data points
 * @param ly
 *   location of y data points
 * @param nptn
 *   number of points on the fitted curve
 * @param lx
 *   location of fitted x points
 * @param lyn
 *   location of fitted y points
 * @param data
 *   working array
 */
void PolyInt(int    npts,
             int     lx,
             int     ly,
             int     nptn,
             int     lx,
             int     lyn,
             double *data)
{
    int    j,j1,i,ix,jx,kx,ixx,jxx;
    double xv,yv,dydx,dydx1,s,y0,dyj,dyj1;

    j  = 1;
    j1 = j+1;
```

1. Stineman, R. W. A consistently well-behaved method of interpolation. *Creative Computing* (July 1980): 54-57.

```

/* Isolate the data(lx+j) and the data(lx+j+1) that bracket xv... */
for (i=1; i<=nptn; i++)
{
    xv = data[lxn+i];
    while (xv > data[lx+j1])
    {
        j++;
        j1 = j+1;
    }

    if (npts == 1)
        yv = data[ly+j];

    if (npts == 2)
        yv = data[ly+2]-(data[lx+j1]-xv)*(data[ly+j1]-data[ly+j])/
            (data[lx+j1]-data[lx+j]);

    if (npts >= 3)
    {
        /*
        thru    * Calculate the slope at the jth point (from fitting a circle
        * 3 points and getting slope of circle).
        */
        ix = 1;
        jx = 2;
        kx = 3;
        if (j != 1)
        {
            ix = j-1;
            jx = j;
            kx = j+1;
        }

        dydx = (((data[ly+jx]-data[ly+ix])*
            (pow(data[lx+kx]-data[lx+jx],2)+
            pow(data[ly+kx]-data[ly+jx],2))+
            (data[ly+kx]-data[ly+jx])*
            (pow(data[lx+jx]-data[lx+ix],2)+
            pow(data[ly+jx]-data[ly+ix],2))))/
            ((data[lx+jx]-data[lx+ix])*
            (pow(data[lx+kx]-data[lx+jx],2)+
            pow(data[ly+kx]-data[ly+jx],2))+
            (data[lx+kx]-data[lx+jx])*
            (pow(data[lx+jx]-data[lx+ix],2)+
            pow(data[ly+jx]-data[ly+ix],2)))));

        if (j == 1)
        {
            ixx = ix;
            jxx = jx;
            s = (data[ly+jxx]-data[ly+ixx])/(data[lx+jxx]-
data[lx+ixx]);

            if (s != 0.0)
            {
                if (!(s >= 0.0 && s > dydx) || (s <= 0.0 && s < dydx))
                    dydx = s+(fabs(s)*(s-dydx))/(fabs(s)+fabs(s-dydx));
                else
                    dydx = 2.0*s-dydx;
            }
        }

        /* Calculate the slope at j+1 point. */
        ix = nptn-2;
        jx = nptn-1;
    }
}

```

```

kx = nptn;

if (j1 != nptn)
{
    ix = j1-1;
    jx = j1;
    kx = j1+1;
}
dydx1 = (((data[ly+jx]-data[ly+ix])*
    (pow(data[lx+kx]-data[lx+jx],2.)+
    pow(data[ly+kx]-data[ly+jx],2.))+
    (data[ly+kx]-data[ly+jx])*
    (pow(data[lx+jx]-data[lx+ix],2.)+
    pow(data[ly+jx]-data[ly+ix],2.)))/
    ((data[lx+jx]-data[lx+ix])*
    (pow(data[lx+kx]-data[lx+jx],2.)+
    pow(data[ly+kx]-data[ly+jx],2.))+
    (data[lx+kx]-data[lx+jx])*
    (pow(data[lx+jx]-data[lx+ix],2.)+
    pow(data[ly+jx]-data[ly+ix],2.))));

if (j1 == nptn)
{
    ixx = jx;
    jxx = kx;
    s = (data[ly+jxx]-data[ly+ixx])/
        (data[lx+jxx]-data[lx+ixx]);
    if (s != 0.0)
    {
        if (!(s >= 0.0 && s > dydx1) ||
            (s <= 0.0 && s < dydx1))
            dydx1 = s+(fabs(s)*(s-dydx1))/(fabs(s)+fabs(s-
dydx1));
        else
            dydx1 = 2.0*s-dydx1;
    }
}

/*
 * Calculate s=slope between j and j+1 points
 * y0 = y-value if linear interp used
 * dyj = delta-y at the j-th point
 * dyj1 = delta-y at the j+1 point
 */
s = (data[ly+j1]-data[ly+j])/(data[lx+j1]-data[lx+j]);
y0 = data[ly+j]+s*(xv-data[lx+j]);
dyj = data[ly+j]+dydx*(xv-data[lx+j])-y0;
dyj1 = data[ly+j1]+dydx1*(xv-data[lx+j1])-y0;

/* Calculate y... */

if (dyj*dyj1 == 0.0)
    yv = y0;
if (dyj*dyj1 > 0.0)
    yv = y0+(dyj*dyj1)/(dyj+dyj1);
if (dyj*dyj1 < 0.0)
    yv = y0+((dyj*dyj1*(xv-data[lx+j]+xv-data[lx+j1]))/
        ((dyj-dyj1)*(data[lx+j1]-data[lx+j])));
}

data[lyn+i] = yv;
}
}

```

Step 2 The XYDataPointsCallback function

Knowing that the PolyInt function uses a single array containing all the raw and calculated independent values, we must prepare this array in the XYDataPointsCallback function and pass it on to the PolyInt function. Once the array is passed on to PolyInt, it will be returned with the calculated points. At that time, we must extract those points from the working array and place them into the array that Tecplot 360 passed to the XYDataPointsCallback function.

The XYDataPointsCallback has the following structure:

1. Allocate and initialize the working array, called Data.
2. Fill the working array with the raw data and the calculated independent values.
3. Pass the working array to the PolyInt function. This will fill in the calculated dependent values.
4. Extract the data from the working array and place into the arrays that Tecplot 360 passed in.
5. Free the working array.

Use the following code for XYDataPointsCallback:

Notice that in this function, the CurveSettings and XYMapNum variables are never referenced.

```
Boolean_t STDCALL XYDataPointsCallback(FieldData_pa RawIndV,
                                       FieldData_pa RawDepV,
                                       CoordScale_e IndVCoordScale,
                                       CoordScale_e DepVCoordScale,
                                       LgIndex_t NumRawPts,
                                       LgIndex_t NumCurvePts,
                                       EntIndex_t XYMapNum,
                                       char *CurveSettings,
                                       double *IndCurveValues,
                                       double *DepCurveValues)
{
    Boolean_t IsOk = TRUE;
    int ii;
    double *Data = NULL;
    int TotalNumDataPts;

    TecUtilLockStart(AddOnID);

    /*
     * Data will contain all the data points and is 1 base:
     * RawIndpts
     * RawDepPts
     * IndCurveValues
     * DepCurveValues
     * Therefore, the array must be large enough to
     * contain all these points: 2*(NumRawPts+NumCurvePts).
     */
    TotalNumDataPts = 2*(NumRawPts+NumCurvePts);
    Data = malloc((TotalNumDataPts+1)*sizeof(double));
    if (Data != NULL)
    {
        /* Initialize Data to contain all zero. */
        for (ii = 0; ii < TotalNumDataPts+1; ii++)
            Data[ii] = 0;
    }
    else
        IsOk = FALSE;

    if (IsOk)
    {
        int lx;
```

```

int ly;
int lxn;
int lyn;

/* Setup the working array, Data. */
PrepareWorkingArray(RawIndV,
                    RawDepV,
                    NumRawPts,
                    NumCurvePts,
                    &lx,
                    &ly,
                    &lxn,
                    &lyn,
                    Data);

/* Perform the curve fit. */
PolyInt(NumRawPts,
        lx,
        ly,
        NumCurvePts,
        lxn,
        lyn,
        Data);

/* Extract the values from Data that were placed there by the
curve fit. */
ExtractCurveValuesFromWorkingArray(NumCurvePts,
                                    lxn,
                                    lyn,
                                    Data,
                                    IndCurveValues,
                                    DepCurveValues);

    free(Data);
}
TecUtilLockFinish(AddOnID);
return IsOk;
}

```

This is because there are no settings which may be configured for this curve-fit. Tecplot 360 requires only a return value (TRUE or FALSE), and that the Ind CurveValues and DepCurveValues arrays are filled. Tecplot 360 will plot any values that are placed in these arrays. If the values do not make sense, the resulting plot will not make sense. The burden is on the add-on writer to make sure that the values placed in these arrays are correct.

Also, notice that there are two functions we have referenced that must still be written. These functions take care of steps 2 and 4 as outlined in the preceding function structure.

Step 3 The PrepareWorkingArray function

This function will fill the working array, Data, with the raw data and the calculated independent curve points. It will also return the indices within the Data array to the different blocks of data. As stated previously:

- **lx** - Start of the raw independent data.
- **ly** - Start of the raw dependent data.
- **lxn** - Start of the calculated independent data.
- **lyn** - Start of the calculated dependent data.

Note the following function above the XYDataPointsCallback function.

```

static void PrepareWorkingArray(FieldData_pa RawIndV,
                               FieldData_pa RawDepV,

```

```

                                LgIndex_t    NumRawPts,
                                LgIndex_t    NumCurvePts,
                                int          *lx,
                                int          *ly,
                                int          *lxn,
                                int          *lyn,
                                double       *Data)
{
    double FirstValidPoint;
    double LastValidPoint;
    double StepSize;
    int    ii;

    /*
     * The following are indices to start points of
     * the data blocks in the 1 based array, Data
     *  lx  - Start of the raw Independent data.
     *  ly  - Start of the raw  Dependent data.
     *  lxn - Start of the calculated independent data.
     *  lyn - Start of the calculated  dependent data.
     *
     * The PolyInt function treats lx,ly,lxn,lyn as 0 base
     * indices, but treats Data as a 1 base array.
     */
    *lx = 0;
    *ly = NumRawPts;
    *lxn = 2*NumRawPts;
    *lyn = 2*NumRawPts+NumCurvePts;

    /* Fill the first blocks of the Data array with the Raw Data Values.
    */
    for (ii = 1; ii <= NumRawPts; ii++)
    {
        Data[*lx+ii] = TecUtilDataValueGetByRef(RawIndV, ii);
        Data[*ly+ii] = TecUtilDataValueGetByRef(RawDepV, ii);
    }

    /*
     * Calculate the size of steps to take while stepping
     * along the independent variable range.
     */
    TecUtilDataValueGetMinMaxByRef(RawIndV,
                                   &FirstValidPoint,
                                   &LastValidPoint);
    StepSize = (LastValidPoint-FirstValidPoint)/(NumCurvePts-1);

    /*
     * Fill the third block of the Data array with the
     * calculated independent values.
     */
    for (ii = 1; ii <= NumCurvePts; ii++)
    {
        double IndV = FirstValidPoint + (ii-1)*StepSize;
        if (IndV > LastValidPoint)
            IndV = LastValidPoint;
        Data[*lxn+ii] = IndV;
    }
}

```

Step 4 The ExtractCurveValuesFromWorkingArray function

This function will extract the calculated data from the working array, Data, and place it in the arrays that were passed to the XYDataPointsCallback function by Tecplot 360. Tecplot 360 will then use these values to plot the curve.

Note the following function above the **XYDataPointsCallback** function.

```
static void ExtractCurveValuesFromWorkingArray(LgIndex_t NumCurvePts,
                                              int          lxn,
                                              int          lyn,
                                              double       *Data,
                                              double       *IndCurveValues,
                                              double       *DepCurveValues)
{
    int ii;
    for (ii = 1; ii <= NumCurvePts; ii++)
    {
        IndCurveValues[ii-1] = Data[lxn+ii];
        DepCurveValues[ii-1] = Data[lyn+ii];
    }
}
```

At this point you should compile the add-on and load it into Tecplot 360. The curve-fit add-on is complete at this point. However, there is other functionality that may be added. In the following sections we will add the probe value callback, and the curve information callback.

Step 5 The ProbeValueCallback function

Because Tecplot 360 will perform a linear interpolation on the points that your curve-fit returns, the ProbeValueCallback function is not required. However, if you have very few points in your curve, the value returned by Tecplot 360's built-in Probe function will return a value that is not on the actual curve, but on the approximated curve.

To avoid this problem, we will write the ProbeValueCallback. This callback will return a value that is actually calculated by your curve-fit. The method we use for this particular curve-fit is outlined following:

The ProbeValueCallback has the following structure:

1. Check that the probed independent value is within the bounds of the raw data.
2. If the number of curve points approximating the curve is small, reassign the number of points approximating the curve to be larger.
3. Allocate and initialize the working array, called Data.
4. Fill the working array with the raw data and the calculated independent values.
5. Insert the probed independent value into the working array, so a curve-fit is done at the actual probed independent value. Save the relative location of this value within the working array.
6. Pass the working array to the PolyInt function. This will fill in the calculated dependent values.
7. Extract the probed dependent value from the working array, using the relative location saved in step 5.
8. Free the working array.

Note the following code in *engine.c*:

```
#define NUMPTSFORPROBING 3000

/**
 * This functions follows a similar process as the
 * XYDataPointsCallback,
 * except it manually inserts ProbeIndValue in the list of the
 * independent curve points. It stores the index in the Data array for
 * that value and uses that relative location to find the calculated
```

```

* ProbeDepValue.
*/
Boolean_t STDCALL ProbeValueCallback(FieldData_pa RawIndV,
                                     FieldData_pa RawDepV,
                                     CoordScale_e IndVCoordScale,
                                     CoordScale_e DepVCoordScale,
                                     LgIndex_t NumRawPts,
                                     LgIndex_t NumCurvePts,
                                     EntIndex_t XYMapNum,
                                     char *CurveSettings,
                                     double ProbeIndValue,
                                     double *ProbeDepValue)
{
    Boolean_t IsOk = TRUE;
    int ii;
    double FirstValidPoint;
    double LastValidPoint;
    double *Data = NULL;
    int TotalNumDataPts;

    TecUtilLockStart(AddOnID);

    /* Make sure the probe is within the bounds of the data. */
    TecUtilDataValueGetMinMaxByRef(RawIndV,
                                    &FirstValidPoint,
                                    &LastValidPoint);
    IsOk = (ProbeIndValue >= FirstValidPoint &&
            ProbeIndValue <= LastValidPoint);

    if (IsOk)
    {
        /*
         * If the Curve has too few points, crank the number of points
         * on the curve up, so we get a good approximation of the curve.
         */
        NumCurvePts = MAX(NUMPTSFORPROBING, NumCurvePts);

        TotalNumDataPts = 2*(NumRawPts+NumCurvePts);
        Data = malloc((TotalNumDataPts+1)*sizeof(double));
        if (Data != NULL)
        {
            /* Initialize Data to contain all zero. */
            for (ii = 0; ii < TotalNumDataPts+1; ii++)
                Data[ii] = 0;
        }
        else
            IsOk = FALSE;
    }

    if (IsOk)
    {
        int lx,ly,lxn,lyn;
        int ProbeValueIndex = -1;

        PrepareWorkingArray(RawIndV,
                            RawDepV,
                            NumRawPts,
                            NumCurvePts,
                            &lx,
                            &ly,
                            &lxn,
                            &lyn,
                            Data);
        IsOk = InsertProbeValueInWorkingArray(ProbeIndValue,

```

```

        NumCurvePts,
        lxn,
        &ProbeValueIndex,
        Data);

    if (IsOk && ProbeValueIndex != -1)
    {
        /* Perform the curve fit. */
        PolyInt(NumRawPts,
                lx,
                ly,
                NumCurvePts,
                lxn,
                lyn,
                Data);
        /* The dependent value is in the same relative location. */
        /* as the probed independent value. */
        *ProbeDepValue = Data[lyn+ProbeValueIndex];
    }
}
if (Data != NULL)
    free(Data);
TecUtilLockFinish(AddOnID);
return IsOk;
}

```

Step 6 The InsertProbeValueInWorkingArray function

This function inserts the probed independent value into the working array so that the curve-fit will be performed exactly at the probed value. This is done by marching through the calculated independent values, and when two values that surround the probed value are found, the probed value replaces the lesser of the two surrounding values in the working array. Also, the relative location of the probed value is saved, so that the calculated dependent value can be extracted from the working array.

Note the following code above the ProbeValueCallback in *engine.c*:

```

static Boolean_t InsertProbeValueInWorkingArray(double
ProbeIndValue,
                                                LgIndex_t NumCurvePts,
                                                int lxn,
                                                int *ProbeValueIndex,
                                                double *Data)
{
    Boolean_t Found = FALSE;
    int ii;

    for (ii = 1; ii < NumCurvePts; ii++)
    {
        /* If the probed value is between the data points record its
        location. */
        if (ProbeIndValue >= Data[lxn+ii] &&
            ProbeIndValue <= Data[lxn+ii+1])
        {
            *ProbeValueIndex = ii;
            Data[lxn+ii] = ProbeIndValue;
            Found = TRUE;
            break;
        }
    }
    return Found;
}

```

Compile and load the add-on into Tecplot 360. Now, you should be able to probe and have a real curve value be returned rather than the linear interpolation computed by Tecplot 360.

Step 7 The CurveInfoStringCallback function

The CurveInfoStringCallback function will pass a string to the **XY-Plot Curve Information** dialog. This string can be any information you wish to present to the dialog. Typical information in this dialogs are the curve coefficients. Since it is beyond the scope of this tutorial to calculate the coefficients of the curve, we will simply present a string to the dialog.

Examine the following code in *engine.c*:

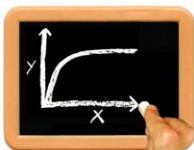
```
Boolean_t STDCALL CurveInfoStringCallback(FieldData_pa RawIndV,
                                          FieldData_pa RawDepV,
                                          CoordScale_e IndVCoordScale,
                                          CoordScale_e DepVCoordScale,
                                          LgIndex_t NumRawPts,
                                          EntIndex_t XYMapNum,
                                          char *CurveSettings,
                                          char **CurveInfoString)
{
    Boolean_t IsOk = TRUE;

    *CurveInfoString = TecUtilStringAlloc(1000, "CurveInfoString");
    strcpy(*CurveInfoString, "Information about the curve goes here.\n");
    strcat(*CurveInfoString, "Such as curve coefficients.");
    return IsOk;
}
```

Again, compile and load the add-on into Tecplot 360. Upon running Tecplot 360, load rainfall.plt¹ and change the curve type to Extended/PolyInt. Now, call up the **XY-Plot Curve Information** dialog. Notice that the string we added is now in the dialog.



As an exercise, add error messages to the XYDataPointsCallback and the ProbeValueCallback functions if they end up returning FALSE. This will inform the user that there was an error.



Example 85: Simple Average Add-on

This tutorial is an example of an extended curve-fit add-on that uses curve settings. The setting that we will be configuring in this add-on, called *SimpAvg*, is the independent variable range. This curve-fit add-on will compute the average of the data within the specified independent variable range. The code in this tutorial is platform-independent.



Please read [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) and/or [Chapter 3: "Creating Add-ons on Windows Platforms"](#) before proceeding with this tutorial.

1. rainfall.plt is included in the \$TEC_360_2013R1/Demo/XY directory on your Tecplot 360 installation, where \$TEC_360_2013R1 is your installation directory. For Windows users, this is typically C:\Program Files\Tecplot\Tec360 2013R1.

Step 1 Getting started



The process described in this manual is the preferred process for creating curve-fit add-ons with configurable settings. Whenever creating an add-on of this type, you should refer to this example as a template.

SimpAvg will use the following source code files. Each one will be automatically created by the `CreateNewAddOn` script (Linux/Macintosh). The project and add-on names will both be *SimpAvg*.

When running `CreateNewAddOn` answer the questions as follows:

- Project Name (Base name) - **SimpAvg**
- Add-on Name - *SimpAvg*
- Company Name - [Your company name]
- Type of Add-on - Extended Curve-Fit
- Language - C
- Allow Configurable Settings - Yes
- Create Callback Function for More Accurate Probing - No

After running the `CreateNewAddOn` script, you should have the following files:

```
engine.c          main.c  guibld.cguicb.c
guidefs.c         ADDGLBL.hENGINE.hGUIDEFS.h
```

You will also have other files specific to your platform, but the files listed are the only ones we will be dealing with. The purpose of each file will be explained in detail as we proceed through the tutorial.

At this point, you should verify that you can compile your add-on and load it into Tecplot 360.

If you are unable to compile or load your add-on, we recommend that you refer to [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) or [Chapter 3: "Creating Add-ons on Windows Platforms"](#) before proceeding.

Step 2 Designing the add-on

Since this curve-fit will have settings which may be configured, we will need to make some decisions before writing the add-on.

1. What are the settings going to be?
 - Use an Independent Variable Range.
 - What is the `IndVarMin`?
 - What is the `IndVarMax`?
2. What are the default settings?
 - `UseIndVarRange` - FALSE.
 - `IndVarMin` - -LARGEDOUBLE (-1E+150).
 - `IndVarMax` - LARGEDOUBLE (1E+150).
3. What is the syntax for the `CurveSettings` string?
 - Newline delimited (spaces delimiting the '=' are required).
 - Example:


```
UseIndVarRange = TRUE\n
IndVarMin = 2\n
IndVarMax = 7\n
```
4. How to maintain the values of the settings?

- The settings string will be maintained by Tecplot 360; however, we will create a struct as follows to hold the values that are contained in the settings string.

```
typedef struct
{
    Boolean_t UseIndVarRange;
    double    IndVarMin;
    double    IndVarMax;
} CurveParams_s;
```

- This structure will be placed in *ADDGLBL.h*.

Step 3 Handling the CurveSettings string

The first thing we will do is lay the groundwork for how to handle the CurveSettings string. Because, this string is maintained by Tecplot 360 and is updated by the add-on, our add-on must be able to parse the string.

We will start with the function that creates the string. This function will make use of the CurveParams_s structure. If you have not done so already, add the CurveParams_s structure, as defined previously, to *ADDGLBL.h*.

Now that the CurveParams_s structure is in place, we will create a function in *engine.c* called *CreateCurveSettingsString*. This function will take one parameter, the CurveParams_s structure, and return a string based on the values of the structure. The function is written as follows:

Add this prototype to *ENGINE.h*:

```
char *CreateCurveSettingsString(CurveParams_s CurveParams);
```

The following code is in *engine.c*:

```
/**
 * Creates a CurveSettings string based on the values
 * in the CurveParams structure that is passed in.
 */
char *CreateCurveSettingsString(CurveParams_s CurveParams)
{
    char S[1000];
    char *CurveSettings;

    if (CurveParams.UseIndVarRange)
        strcpy(S, "UseIndVarRange = TRUE\n");
    else
        strcpy(S, "UseIndVarRange = FALSE\n");

    sprintf(&S[strlen(S)], "IndVarMin = %G\n", CurveParams.IndVarMin);
    sprintf(&S[strlen(S)], "IndVarMax = %G\n", CurveParams.IndVarMax);

    S[strlen(S)] = '\0';
    CurveSettings = TecUtilStringAlloc(strlen(S), "CurveSettings");
    strcpy(CurveSettings, S);
    return CurveSettings;
}
```

Notice that this function calls *TecUtilStringAlloc*. The calling function is responsible for de-allocating the string returned by *CreateCurveSettingsString*. Also notice that the string is newline delimited, as discussed previously.

Now that we have a function that creates the CurveSettings string, create a function that will parse the newline delimited string and populate the CurveParams_s structure. The function that we will be writing will use several convenient functions that are defined in *adkutil.h*. Simply, add the line:

```
#include "ADKUTIL.h"
```

at the top of *engine.c*.

The function that parses the CurveSettings string will take three parameters:

- **XYMapNum**, - the XY-map that is currently being operated on.
- CurveSettings string.
- a pointer to the CurveParams_s structure. This function will not only parse the CurveSettings string, but also repair the string if the syntax is incorrect.

The following code is in *engine.c*:

```
/**
 * This function makes use of functions found in the
 * adkutil.c module to parse the CurveSettings string.
 */
void GetValuesFromCurveSettings(EntIndex_t XYMapNum,
                                char *CurveSettings,
                                CurveParams_s *CurveParams)
{
    Boolean_t IsOk = TRUE;
    # define MAXCHARS 50
    char Command[MAXCHARS+1];
    char ValueString[MAXCHARS+1];
    char *CPtr;
    char *ErrMsg = NULL;

    if (CurveSettings != NULL && strlen(CurveSettings) > 0)
    {
        CPtr = CurveSettings;
        while (IsOk && *CPtr)
        {
            if (GetArgPair(&CPtr,
                          Command,
                          ValueString,
                          MAXCHARS,
                          &ErrMsg))
            {
                if (Str_ustrcmp(Command, "USEINDVARRANGE") == 0)
                {
                    Boolean_t UseRange;
                    IsOk = Macro_GetBooleanArg(Command,
                                              ValueString,
                                              &UseRange,
                                              &ErrMsg);

                    if (IsOk)
                        CurveParams->UseIndVarRange = UseRange;
                }
                else if (Str_ustrcmp(Command, "INDVARMIN") == 0)
                {
                    double Min;
                    IsOk = Macro_GetDoubleArg(Command,
                                              ValueString,
                                              -LARGEDOUBLE,
                                              LARGEDOUBLE,
                                              &Min,
                                              &ErrMsg);

                    if (IsOk)
                        CurveParams->IndVarMin = Min;
                }
                else if (Str_ustrcmp(Command, "INDVARMAX") == 0)
                {

```

```

        double Max;
        IsOk = Macro_GetDoubleArg(Command,
                                   ValueString,
                                   -LARGEDOUBLE,
                                   LARGEDOUBLE,
                                   &Max,
                                   &ErrMsg);

        if (IsOk)
            CurveParams->IndVarMax = Max;
        }
    else
    {
        ErrMsg = TecUtilStringAlloc((strlen(Command)+100),
                                     "error message");
        sprintf(ErrMsg, "Unknown argument: %s.", Command);
        IsOk = FALSE;
    }
}
else /* GetArgPair Failed. */
    IsOk = FALSE;
}
}
else /* CurveSettings is an invalid string. */
    IsOk = FALSE;

/* Repair the string. Display the Error Message if needed. */
if (!IsOk)
{
    char *NewCurveSettings = NULL;
    InitializeCurveParams(CurveParams);
    NewCurveSettings = CreateCurveSettingsString(*CurveParams);

    if (NewCurveSettings != NULL)
    {
        TecUtilCurveSetExtendedSettings(XYMapNum, NewCurveSettings);
        TecUtilStringDealloc(&NewCurveSettings);
    }
    if (ErrMsg != NULL)
    {
        TecUtilDialogErrMsg(ErrMsg);
        TecUtilStringDealloc(&ErrMsg);
    }
}
}
}

```

Notice at the bottom of this function we repair the CurveSettings string if it was invalid. It could be that the syntax was wrong, or that the string had not yet been initialized. Either way, we call the function `InitializeCurveParams` in which we setup the `CurveParams_s` structure with default values. Then, we create a new CurveSettings string, which is constructed with the default values. Finally, we set the CurveSettings string for the current XY-map, `XYMapNum`, by calling `TecUtilCurveSetExtendedSettings`.

Step 4 The InitializeCurveParams function

Examine the following code in *engine.c*:

```

void InitializeCurveParams(CurveParams_s *CurveParams)
{
    CurveParams->UseIndVarRange = FALSE;
    CurveParams->IndVarMin      = -LARGEDOUBLE;
    CurveParams->IndVarMax      = LARGEDOUBLE;
}

```

Now that we have the laid groundwork for handling the CurveSettings string, we can move on to creating the rest of the add-on.

Step 5 Registering the add-on with Tecplot 360

The first thing that must happen when the add-on is loaded into Tecplot 360 is that it must be registered. In *main.c* there is a function:

This function will register the curve-fit add-on with Tecplot 360. Notice that parameter three is NULL.

```
TecUtilCurveRegisterExtCrvFit(ADDON_NAME,
                              XYDataPointsCallback,
                              NULL, /* ProbeValueCallback */
                              CurveInfoStringCallback,
                              CurveSettingsCallback,
                              AbbreviatedSettingsStringCallback);
```

This is because we are not adding the ProbeValueCallback.

At this point verify that the add-on will compile and load into Tecplot 360.

Step 6 Creating the Dialog

In this step we will create the dialog that will be displayed when the user selects the [Curve Settings] button on the Curves page of the **Mapping Style** (when the Curve Type is of type *SimpAvg*). *We strongly recommend that the curve-fit dialog be modal.*

The dialog will have five controls: one toggle, two text fields, and two labels.

1. Because we are using [Tecplot GUI Builder](#) (TGB), the dialog template is the Tecplot 360 layout file *gui.lay* (located in your project directory). Load *gui.lay* into Tecplot 360, select Tecplot GUI Builder from the **Tools** menu.



If **Tecplot GUI Builder** is not available from the **Tools** menu in Tecplot 360, refer to [Section 25 - 1 "Using Tecplot GUI Builder"](#) for instructions are running Tecplot 360 with the GUI Builder.

2. Edit the layout as follows:

3. There will be callbacks associated with each of the text fields, and the toggle button. By default, the TGB adds generic variable names for these controls. Change these variable names by double-clicking on the control and selecting the [Options] button in the resulting **Text Details** dialog. The variable name can be changed in the Macro Function field of the **Text Options** dialog.
 - a. Double-click on the Use Independent Variable Range toggle and select Options. In the Macro Function field, type `VarName=UseIndVarRange`. This will give the callback a meaningful name.
 - b. Appropriate names for the text fields are `IndVarMin` and `IndVarMax`. Although we will not be performing any operations in the text field callbacks, giving them meaningful names is recommended. Set the label names to `"VarName=Min"` and `"VarName=Max"`.

4. Now, double-click on the dialog frame and verify that the frame name is as follows:

```
ID=1 MODE=MODAL TITLE="Simple Average"
```

5. You can now build the source for this layout. Select Go Build from the **GUI Builder** dialog.
6. Now that TGB has created new stub files, be sure to copy the toggle and text field callbacks from *guibc.tmp* into *guibc.c*.

Step 3 Launching and Initializing the Dialog

The add-on dialog is launched by the `CurveSettingsCallback` function in *engine.c*. The parameter `XYMapSet` is the set of XY-maps that were selected in the **Mapping Style** dialog at the time Curve Settings was selected. The parameter `XYMapSettings` is a string list containing the `CurveSettings` strings of all the XY-maps in the set, `XYMapSet`.

When the [Curve Settings] is selected, the function `CurveSettingsCallback` is called by Tecplot 360. In this function we will save the `XYMapSet` and `XYMapSettings` so we can use them later in the *guibc.c* module. These variables are needed in *guibc.c* in order to properly initialize the dialog fields.

In *engine.c* verify that `CurveSettingsCallback` is as follows:

```
void STDCALL CurveSettingsCallback(Set_pa      XYMapSet,
                                  StringList_pa XYMapSettings)
{
    TecUtilLockStart(AddOnID);
    /*
     * Save off XYMapSettings and SelectedXYMaps for use
     * in the functions in guibc.c
     */
    GlobalCurve.XYMapSet      = XYMapSet;
    GlobalCurve.XYMapSettings = XYMapSettings;

    /* Build and Launch the dialog */
    BuildDialog1(MAINDIALOGID);
    TecGUIDialogLaunch(Dialog1Manager);

    TecUtilLockFinish(AddOnID);
}
```

`GlobalCurve` is a global structure that maintains the curve settings when the dialog is launched. This structure must be declared in `ENGINE.h` as follows:

```
typedef struct
{
    StringList_pa XYMapSettings;
    Set_pa        XYMapSet;
} GlobalCurve_s;
```

Now declare the variable `GlobalCurve` in *engine.c*. Immediately below the `#include` statements in *engine.c* and *guibc.c*, type the following:

```
GlobalCurve_s GlobalCurve;
```

Finally, make sure the line:

```
#include "ENGINE.h"
```

exists in *guibc.c*.

Step 4 Initializing the Dialog

Initialization of the dialog is taken care of in *guibc.c* in the function `Dialog1Init_CB`. When initializing the dialog, we must place the correct values into each field, and we must also set the sensitivities of each field. In the case of this dialog the sensitivities are as follows:

- **UseIndVarRange** - Toggle, always active.
- **IndVarMin** - Text field, active when `UseIndVarRange` is checked.

- **IndVarMax** - Text field, active when UseIndVarRange is checked.
- **Min** - Label, active when UseIndVarRange is checked.
- **Max** - Label, active when UseIndVarRange is checked.

To set the sensitivities we create the following function in *guicb.c*. Be sure this function is placed above the *Dialog1Init_CB* function:

If only one XY-map is selected, the XYMapSettings string list will have only one member, and that member

```
static void UpdateMainDialogSensitivities(void)
{
    Boolean_t Sensitive = TecGUIToggleGet(UseIndVarRan_TOG_D1);
    TecGUISetSensitivity(IndVarMin_TF_D1, Sensitive);
    TecGUISetSensitivity(IndVarMax_TF_D1, Sensitive);
    TecGUISetSensitivity(Min_LBL_D1, Sensitive);
    TecGUISetSensitivity(Max_LBL_D1, Sensitive);
}
```

will be the CurveSettings for that mapping. However, when there is more than one mapping selected, and they have different curve settings, how do we decide to initialize the fields on the dialog? Use the following method:

- If all mappings have the same values for any particular field, that value will be used.
- If the selected mappings have different values for any particular field, the default value is used.

To help initialize the fields, we will create a function that will determine the proper value for each variable. The function will then return the appropriate value: the default value if the maps have different settings for that value, or the value that is set if all maps have the same setting for that value. The function is defined as follows.

The following function is in *guicb.c*:

```
static void InitializeGUICurveParams(CurveParams_s *CurveParamsPtr)
{
    char *CurveSettings = NULL;
    CurveParams_s OrigCurveParams;
    Boolean_t UseIndVarRangeIsSame = TRUE;
    Boolean_t IndVarMinIsSame = TRUE;
    Boolean_t IndVarMaxIsSame = TRUE;
    int ii;
    int NumMembers;

    /* Get the CurveParams associated with the first mapping. */
    CurveSettings = TecUtilStringListGetString(GlobalCurve.XYMapSettings,
    1);
    GetValuesFromCurveSettings(
    (EntIndex_t)TecUtilSetGetNextMember(GlobalCurve.XYMapSet, TECUTILSETNOT
    MEMBER),
    CurveSettings,
    &OrigCurveParams);
    if (CurveSettings != NULL)
        TecUtilStringDealloc(&CurveSettings);

    NumMembers = TecUtilStringListGetCount(GlobalCurve.XYMapSettings);

    /*
    * Compare the value of the first mapping with all the other mappings.
    * This loop will not be done if there is only one mapping selected.
    */
}
```

```

    for (ii = 2; ii <= NumMembers; ii++)
    {
        CurveParams_s TmpParams;
        CurveSettings =
        TecUtilStringListGetString(GlobalCurve.XYMapSettings, ii);
        GetValuesFromCurveSettings(
        (EntIndex_t)TecUtilSetGetNextMember(GlobalCurve.XYMapSet, ii),
        CurveSettings,
        &TmpParams);
        if (UseIndVarRangeIsSame)
            UseIndVarRangeIsSame = (TmpParams.UseIndVarRange ==
            OrigCurveParams.UseIndVarRange);

        if (IndVarMinIsSame)
            IndVarMinIsSame = (TmpParams.IndVarMin ==
            OrigCurveParams.IndVarMin);

        if (IndVarMaxIsSame)
            IndVarMaxIsSame = (TmpParams.IndVarMax ==
            OrigCurveParams.IndVarMax);

        if (CurveSettings != NULL)
            TecUtilStringDealloc(&CurveSettings);
    }
    /*
    * Initialize the CurveParamsPtr to the default values.
    * If all mappings have the same value for a particular parameter,
    * use that value instead.
    */
    InitializeCurveParams(CurveParamsPtr);

    if (UseIndVarRangeIsSame)
        CurveParamsPtr->UseIndVarRange = OrigCurveParams.UseIndVarRange;
    if (IndVarMinIsSame)
        CurveParamsPtr->IndVarMin = OrigCurveParams.IndVarMin;
    if (IndVarMaxIsSame)
        CurveParamsPtr->IndVarMax = OrigCurveParams.IndVarMax;
}

```

Finally we will add a function to initialize the dialog fields, which will be called from the `Dialog1Init_CB` function, as described following. This function also calls `InitializeGUICurveParams`, which was previously defined.

The following function is in *guicb.c* below the `UpdateMainDialogSensitivities` and below the `InitializeGUICurveParams` function:

Compile and run your add-on to make sure that the fields and sensitivities are initialized correctly. The

```

static void UpdateMainDialog(void)
{
    CurveParams_s CurveParams;
    InitializeGUICurveParams(&CurveParams);
    TecGUIToggleSet(UseIndVarRan_TOG_D1, CurveParams.UseIndVarRange);
    TecGUITextFieldSetDouble(IndVarMin_TF_D1, CurveParams.IndVarMin, "%G");
    TecGUITextFieldSetDouble(IndVarMax_TF_D1, CurveParams.IndVarMax, "%G");
    UpdateMainDialogSensitivities();
}

```

dialog should appear with the “Use Independent Variable Range” toggle off, and the remaining controls should be insensitive. Using the “Use Independent Variable Range” toggle will not change the sensitivities of the dialog at this point.

Step 5 Making the Dialog Operational

To make the dialog fully operational, there are two things that must be done. The first is to update the sensitivities of the text field controls when **Use Independent Variable Range** is toggled-on. The second is to make the dialog set the values when [OK] is selected.

1. **Updating the Sensitivities** - To be sure that the text field sensitivities are updated when the toggle button is selected, include the following code:

```
static void UseIndVarRan_TOG_D1_CB(const int *I)
{
    TecUtilLockStart(AddOnID);
    /*
     * Make sure to update the sensitivities when
     * the toggle button is pressed.
     */
    UpdateMainDialogSensitivities();
    TecUtilLockFinish(AddOnID);
}
```

2. The process to follow when [OK] is selected is:
 - a. Collect the information from the dialog.
 - b. Create a new CurveSettings string.
 - c. Call TecUtilXYMapSetCurve with the appropriate parameters to set the extended curve settings for the set of XY-maps.
 - d. Drop the dialog.

The following function collects the information from the dialog and places it into the CurveParams structure:

The Dialog1OkButton_CB function to look as follows:

```
static void AssignCurveParams(CurveParams_s *CurveParams)
{
    CurveParams->UseIndVarRange = TecGUIToggleGet(UseIndVarRan_TOG_D1);
    /*
     * Note this function returns a boolean alerting user whether or not
     * input value is legitimate. Some error checking may be added here.
     */
    TecGUITextFieldGetDouble(IndVarMin_TF_D1,&CurveParams->IndVarMin);
    TecGUITextFieldGetDouble(IndVarMax_TF_D1,&CurveParams->IndVarMax);
}
```

At this point, the dialog should be fully functional. The dialog will be initialized with the correct values

```
static void Dialog10kButton_CB(void)
{
    /* Only unlock tecplot here because a modal dialog was launched. */
    /* When curve settings change, Tecplot must be informed of the change. */
    /*
    char *CurveSettings = NULL;
    CurveParams_s CurveParams;

    /* Assign the new curve parameters from the dialog settings. */
    AssignCurveParams(&CurveParams);

    /* Create the Curve Settings string from the new curve parameters. */
    CurveSettings = CreateCurveSettingsString(CurveParams);
    if (CurveSettings != NULL)
    {
        EntIndex_t Map;
        TecUtilSetForEachMember(Map, GlobalCurve.XYMapSet)
        {
            TecUtilCurveSetExtendedSettings(Map, CurveSettings);
        }
        TecUtilStringDealloc(&CurveSettings);
    }

    TecGUIDialogDrop(Dialog1Manager);
    TecUtilLockFinish(AddOnID);
}
```

and sensitivities. The sensitivities will be updated correctly, and Tecplot 360 will be informed when the CurveSettings string is changed.

Step 5 Updating the Mapping/Zone Style Dialog

To update the **Mapping Style** dialog, we move back to the *engine.c* module. The CurveSettings field of the Mapping/Zone Style dialog will be filled with the string returned by the AbbreviatedSettingsStringCallback function. If this function is undefined, or returns a value of NULL, the CurveSettings string that Tecplot 360 stores will be used in the **Mapping Style** dialog.

To create this string, we will evaluate the CurveSettings string and create a legible output string. The string we will produce will look like:

- If using the Independent Variable Range, IndVarMin = 2 and IndVarMax = 7:
"IndVarRange: Min = 2; Max = 7"
- If not using the Independent Variable Range:
"No IndVarRange"

```
void STDCALL AbbreviatedSettingsStringCallback(EntIndex_t XYMapNum,
                                              char *CurveSettings,
                                              char
**AbbreviatedSettings)
{
    CurveParams_s CurveParams;
    char *S;

    TecUtilLockStart(AddOnID);
    GetValuesFromCurveSettings(XYMapNum,
                              CurveSettings,
                              &CurveParams);
```

```

S = TecUtilStringAlloc(80, "Abbreviated Settings");

if (CurveParams.UseIndVarRange)
{
    sprintf(S,
            "IndVar Range: Min = %G; Max = %G",
            CurveParams.IndVarMin,
            CurveParams.IndVarMax);
    *AbbreviatedSettings = S;
}
else
{
    strcpy(S, "No IndVarRange");
    *AbbreviatedSettings = S;
}
TecUtilLockFinish(AddOnID);
}

```

Compile the add-on and verify that you can change the settings via your dialog, and that settings are displayed on the **Mapping Style** dialog.

Step 6 The XYDataPointsCallback

We will need to alter the XYDataPointsCallback to determine the proper independent variable range. This range is the range limited by the extents of the data and the values specified in the **Curve-Fit** dialog. Alter the XYDataPointsCallback as follows:

```

Boolean_t STDCALL XYDataPointsCallback(FieldData_pa RawIndV,
                                       FieldData_pa RawDepV,
                                       CoordScale_e IndVCoordScale,
                                       CoordScale_e DepVCoordScale,
                                       LgIndex_t NumRawPts,
                                       LgIndex_t NumCurvePts,
                                       EntIndex_t XYMapNum,
                                       char *CurveSettings,
                                       double *IndCurveValues,
                                       double *DepCurveValues)
{
    Boolean_t IsOk = TRUE;

    int ii;
    double Average;
    double Delta = 0.0;
    double IndVarMin,
    IndVarMax;
    CurveParams_s CurveParams;

    TecUtilLockStart(AddOnID);

    /* Get the min and max values of the independent variable. */
    TecUtilDataValueGetMinMaxByRef(RawIndV,
                                   &IndVarMin,
                                   &IndVarMax);

    /* Get the curve parameters */
    GetValuesFromCurveSettings(XYMapNum,
                              CurveSettings,
                              &CurveParams);

    if (CurveParams.UseIndVarRange)
    {
        /*
         * Adjust the independent variable range to fall either within
         * the range of data or the range specified by the

```

```

        * CurveParams structure.
        */
        IndVarMin = MAX(IndVarMin, CurveParams.IndVarMin);
        IndVarMax = MIN(IndVarMax, CurveParams.IndVarMax);
    }

    Delta = (IndVarMax-IndVarMin)/(NumCurvePts-1);

    /*
     * Find the average value of the raw dependent variable for the
     * default curve fir (straight line at average).
     */
    Average = SimpleAverage(RawDepV,
                           RawIndV,
                           NumRawPts,
                           IndVarMin,
                           IndVarMax);

    /*
     * Step through all the points along the curve and set the
     * DepCurveValues to the Average at each IntCurveValue.
     */
    for (ii = 0; ii < NumCurvePts; ii++)
    {
        IndCurveValues[ii] = ii*Delta + IndVarMin;
        DepCurveValues[ii] = Average;
    }

    TecUtilLockFinish(AddOnID);
    return IsOk;
}

```

Notice that the SimpleAverage function has also been changed. We are now passing more information to the SimpleAverage function so it can make the decision about what points to include in the average value calculation. Alter the SimpleAverage function as follows:

```

/**
 * Function to compute the average of the raw dependent variable for the
 * default fit (straight line at average).
 *
 * REMOVE THIS FUNCTION FOR OTHER FITS.
 */
double SimpleAverage(FieldData_pa RawDepV,
                    FieldData_pa RawIndV,
                    LgIndex_t NumRawPts,
                    double IndVarMin,
                    double IndVarMax)
{
    int ii;
    int Count = 0;
    double Sum = 0;

    for (ii = 0; ii < NumRawPts; ii++)
    {
        double IndV = TecUtilDataValueGetByRef(RawIndV, ii+1);

        /*
         * Only compute the average on values that fall in the
         * specified range of the independent variable.
         */
        if ( IndV >= IndVarMin && IndV <= IndVarMax)
        {
            Sum += TecUtilDataValueGetByRef(RawDepV, ii+1);
        }
    }
}

```

```

        Count++;
    }
}

return (Sum/Count);
}

```

The SimpleAverage function is also used in the CurveInfoStringCallback so we will have to alter that function as well. You will notice that the process in CurveInfoStringCallback is very similar to the process used in XYDataPointsCallback. The CurveInfoStringCallback function looks as follows:

```

Boolean_t STDCALL CurveInfoStringCallback(FieldData_pa RawIndV,
                                          FieldData_pa RawDepV,
                                          CoordScale_e IndVCoordScale,
                                          CoordScale_e DepVCoordScale,
                                          LgIndex_t NumRawPts,
                                          EntIndex_t XYMapNum,
                                          char *CurveSettings,
                                          char **CurveInfoString)
{
    Boolean_t IsOk = TRUE;
    CurveParams_s CurveParams;
    double IndVarMin, IndVarMax;
    double Average;

    TecUtilLockStart(AddOnID);

    /*
     * If this function is not registered with Tecplot, no curve
     * information will be displayed in the XY-Curve Info dialog.
     */
    *CurveInfoString = TecUtilStringAlloc(30, "CurveInfoString");

    /* Get the curve parameters. */
    GetValuesFromCurveSettings(XYMapNum, CurveSettings, &CurveParams);

    if (CurveParams.UseIndVarRange)
    {
        /*
         * Adjust the Independent variable range to fall either within
         * the range of the data or the range specified by the
         * CurveParams structure.
         */
        IndVarMin = CurveParams.IndVarMin; /* initialize these values */
        IndVarMax = CurveParams.IndVarMax;
        IndVarMin = MAX(IndVarMin, CurveParams.IndVarMin);
        IndVarMax = MIN(IndVarMax, CurveParams.IndVarMax);
    }

    Average = SimpleAverage(RawDepV,
                            RawIndV,
                            NumRawPts,
                            IndVarMin,
                            IndVarMax);

    sprintf(*CurveInfoString, "Average is: %G\n", Average);

    TecUtilLockFinish(AddOnID);
    return IsOk;
}

```

The add-on is now complete. You should compile the add-on at this time and verify that it works as expected.



As a further exercise, add error-checking to the dialog so that the minimum value is greater than the maximum value.

Creating a Data Converter

A data set converter is the easier of the two data reader types to write. The primary component of a data set converter is a function that reads in a non-Tecplot 360 data format and writes out a binary Tecplot 360 data file. The ADK provides functions that make it easy to write out a Tecplot 360 binary data file once you have read in your own data. A data set converter does not require graphical user interface. The standard Tecplot 360 file dialogs are used and Tecplot 360 manages the reading in of multiple files, partial reads, and so on. Data set converters are registered with Tecplot 360 by making the following call in `InitTecAddon`:

```
TecUtilImportAddConverter(ConverterCallback,
                          "MyConverterName",
                          "FNameExtension");
```

- *ConverterCallback* is the function that converts the data. It is in *engine.c*, and has these parameters:

```
Boolean_t ConverterCallback(char *DataFName,
                           char *TempBinFName,
                           char **MessageString);
```

ConverterCallback reads from the file *DataFName*, writes to the file *TempBinFName*, and if and only if there are any errors, places the error messages in the string *MessageString*. *MessageString* must be allocated inside *ConverterCallback* using `TecUtilStringAlloc` as follows:

```
*MessageString = TecUtilStringAlloc(Size, "Error message");
```

You can use `strcpy` to assign an error string. For example:

```
strcpy(*MessageString, "My Error");
return FALSE;
```

- *MyConverterName* is a unique name assigned to your data set converter. It must be less than 32 characters long. This is also used as the text in the **Load Data File(s)** dialog in the Tecplot 360 interface.
- *FNameExtension* is the extension you want as the default for the file dialog when it is displayed by Tecplot 360.

A data set converter should use *only* the following functions to write out the binary Tecplot 360 data file:

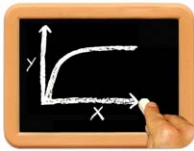
```
TecUtilTecAux
TecUtilTecDat
TecUtilTecEnd
```

```

TecUtilTecFace
TecUtilTecFil
TecUtilTecForeign
TecUtilTecGeo
TecUtilTecGeoX
TecUtilTecIni
TecUtilTecLab
TecUtilTecNod
TecUtilTecNode
TecUtilTecTxt
TecUtilTecTxtX
TecUtilTecUsr
TecUtilTecVAux
TecUtilTecZAux
TecUtilTecZne
TecUtilTecZneX

```

These functions duplicate the capabilities of the TECIO functions described in [Section 3 - 7 “Binary Data File Function Reference” in the Data Format Guide](#).



Example 86: Simple Spreadsheet Converter

This tutorial illustrates how to create an add-on, called *Converter*. The *Converter* add-on is an example of how to load a comma- or space-delimited list of values into Tecplot 360. When you run Tecplot 360 after you have completed this tutorial, *Converter* will appear under the **Load Data File(s)** option of Tecplot 360's **File** menu.

All of the examples of the source code shown in this tutorial are included in the Tecplot 360 distribution in `$TEC_360_2013R1/adk/samples/cnvss`.

Step 1 Add-on setup

Converter uses source code files created by the `CreateNewAddOn` script (Linux/Macintosh). Our project name will be “Converter” and the add-on name will be “Simple Spreadsheet Converter.”

When running `CreateNewAddOn`, answer the questions as follows:

- Project name (base name) - Converter
- Add-on name - Simple Spreadsheet Converter
- Company Name - [Your company name]
- Type of add-on - Data Converter
- Language - C
- Use TGB to create a platform-independent GUI? - No
- Add a menu callback to the Tecplot? - No

After running `CreateNewAddOn`, you should have the following files:

- **engine.c, ENGINE.h** - The files *ENGINE.h* and *engine.c* contain the main converter function. *engine.c* currently has a short message saying that the converter is under construction. Throughout this tutorial, code will be added to *engine.c* so when Tecplot 360 calls the `ConverterCallback` function it will perform of loading the file.

- **main.c** - contains a function called `InitTecAddOn`. This registers the add-on with Tecplot 360. Note that within this function there are other function calls which tell Tecplot 360 the name of the add-on and state that it is a converter. The `InitTecAddOn` function is called by Tecplot 360 exactly when the add-on is first loaded, and is not called again.
- **ADDGLBL.h** - contains information specific to the add-on, such as its name, version number, and date

There will be other files specific to your platform, however, we will only be dealing with those listed. Verify that the add-on will compile and that it can be loaded into Tecplot 360. If any problems are encountered, refer to [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#).

Step 2 Modifying the ConverterCallback function

When *Converter* is loaded by Tecplot 360, an option called Simple Spreadsheet Converter will appear in the **Load Data File(s)** dialog accessed from Tecplot 360's **File** menu. When *Converter* is launched, Tecplot 360 will ask for a file to convert. *Converter* passes this file name to the `ConverterCallback` function. Tecplot 360 will also create a unique temporary file name and pass that to `ConverterCallback` as well.

In `ConverterCallback` we are required to:

- Open the file `DataFName`.
- Convert the data and create a Tecplot 360 binary data file.
- Close the file `DataFName`.
- Inform Tecplot 360 if there were any errors.

Note how the `ConverterCallback` function satisfies these requirements:

```
char *TempBinFName,
char **MessageString)
{
    Boolean_t IsOk = TRUE;
    FILE *f;

    TecUtilLockStart(COMMANDPROCESSORID);
    /* If there is no error, remember to free MessageString. */
    *MessageString = TecUtilStringAlloc(1000, "MessageString for CNVSS");
    /* Try to open the file. */
    f = fopen(DataFName, "rb");

    /* Make sure the file was opened. */
    if (!f)
    {
        strcpy(*MessageString, "Cannot open input file.");
        IsOk = FALSE;
    }
    /* Do the conversion. */
    if (IsOk)
        IsOk = DoConversion(f, TempBinFName, MessageString);

    /* Close the file. */
    fclose(f);

    /* If there was no errors, deallocate MessageString. */
    if (IsOk)
        TecUtilStringDealloc(MessageString);

    TecUtilLockFinish(COMMANDPROCESSORID);
    return IsOk;
}
```

This function does the following:

- **Creates an error message** - *MessageString is allocated here because the DoConversion function (which will be explained later) may alter the error message that is reported.
- **Attempts to open the file** - If the file cannot be opened, it sets IsOk to FALSE, and resets the *MessageString to reflect the fact that the file could not be opened.
- **If the file was opened, it converts it** - The task of conversion is handed off to the DoConversion function.
- Some clean up is performed, such as closing the file, de-allocating *MessageString if there were no errors, and returning IsOk. If IsOk is FALSE at the end of the function, there was an error. Tecplot 360 will use the string in *MessageString to display an error message.

Step 3 Writing the DoConversion Function

Now that the file is open, we want to perform the conversion. In ConverterCallback the job of performing the conversion is passed to the DoConversion function. DoConversion is responsible for parsing the file to be converted and sending specific information to the TecUtil¹ functions. In writing the DoConversion function, we are going to make the following assumptions about the format of the incoming file:

- the variables are at the top of the file, contained in quotes, and separated by commas or spaces
- the data follows the variables and is separated by commas or spaces

An example of such a file would be:

```
"Var 1" "Var 2" "Var 3"
1.23, 4.4, 3.24
2.45, 3.56, 5.2
3.2, 2.15, 7.56
```

The basic form of a conversion function is:

- Get variable names from the file into a comma-separated string.
- Call TecUtilTecIni to initialize the temporary file.
- Call TecUtilTecZne to add a zone.
- Get data points into an array.
- Call TecUtilTecDat to add the data points to the temporary file.
- Call TecUtilTecEnd to close the temporary file.

Our converter will perform other function, in addition to the minimum requirements listed here. There are two functions used for parsing the income file. These are GetVars and get_token. GetVars takes two parameters: a FILE* and a StringList_pa. Be sure that you understand the StringList_pa data type before continuing. For a discussion of StringList_pa see the [Chapter 17: "String Lists"](#). GetVars will parse the text file for the variable names and place them in the string list. get_token takes a FILE* and will parse a text file for items which are separated by commas or spaces. There is no checking to make sure that the item is a valid number. get_token will update a global variable called _token, which is used in DoConversion:

```
static Boolean_t DoConversion(FILE *f,
                             char *TempFName,
                             char **MessageString)
{
    Boolean_t IsOk = TRUE;
    StringList_pa VarList = TecUtilStringListAlloc(); /* Variable list.
*/
    int i;
    int NumValues;
    int NumVars;
    int IMax;
```

1. A discussion of the TecUtil functions is available in the [ADK Reference Manual](#).

```

/* First, we need to read all of the variables. */
GetVars(f,VarList);

/* Make sure there is at least one variable. */
if (IsOk && TecUtilStringListGetCount(VarList) < 1)
{
    strcpy(*MessageString,"No variables defined.");
    IsOk = FALSE;
}

if (IsOk)
{
    /* Debug and VIsDouble are flags used by TecUtilTecIni(). */
    int Debug = 0;
    int VIsDouble = 1;

    /* Set JMax and KMax to 1 because we are creating an. */
    /* I-ordered data set. */

    int JMax=1,KMax=1;
    char VarNames[5000];
    char *s;

    NumValues = 0;

    /* VarList was filled by the function GetVars. */
    NumVars = TecUtilStringListGetCount(VarList);

    /* Count the number of data points. */
    while (get_token(f))
    {
        NumValues++;
    }

    /*
    * Get_token() changed where the file pointer is pointing, so
    * we must rewind to the start of the data.
    */
    fseek(f,&_DataStartPos);

    /* Compute the number of data points. */
    IMax = NumValues/NumVars;

    /* FillVarNames with the variable names in VarList. */
    strcpy(VarNames,"");
    for (i=1; i<=NumVars && IsOk; i++)
    {
        s = TecUtilStringListGetString(VarList,i);
        strcat(VarNames,s);
        if (i<NumVars)
            strcat(VarNames,",");
        TecUtilStringDealloc(&s);
    }

    /*
    * Use the TecUtilTecIni() function to initialize the TempFName
    * file and fill it with the data set title and the variable name.
    */
    if (TecUtilTecIni("ConvertedDataset", VarNames,
                    TempFName,"",&Debug,&VIsDouble) != 0)
    {
        strcpy(*MessageString,"Could not create data set.");
        IsOk = FALSE;
    }
}

```

```

/*
 * Use TecUtilTecZne to add the first zone.
 * In this case, it is the only zone.
 */
if (IsOk && TecUtilTecZne("Zone 1",
                        &IMax,&JMax,&KMax,
                        "POINT",NULL) != 0)
{
    strcpy(*MessageString,"Could not add zone.");
    IsOk = FALSE;
}

/* Now add the data. */
if (IsOk)
{
    LgIndex_t PointIndex = 1;
    int Skip = 0;

    /* Allocate space to temporarily store the values. */
    double *LineValues = (double*)
calloc(NumValues,sizeof(double));

    /* Get the values into the array LineValues. */
    for (i=0; i<NumValues; i++)
    {
        get_token(f);
        LineValues[i] = atof(_token);
    }

    /*
     * Use the function TecUtilTecDat() to fill the
     * temporary file with the values stored in the LineValues.
     */
    if (TecUtilTecDat(&NumValues,(void*)LineValues,&VIsDouble) !=
0)
    {
        strcpy(*MessageString,"Error loading data.");
        IsOk = FALSE;
    }

    /* Free LineValues now that we are done using it. */
    free(LineValues);
}

/* Calling TecUtilTecEnd() closes the temporary file. */
if (TecUtilTecEnd() != 0)
{
    IsOk = FALSE;
    strcpy(*MessageString,"Error closing temporary file, "
"could not create data set.");
}

TecUtilStringListDealloc(&VarList);
return IsOk;
}

```

Step 4 Parsing the code

A discussion of the parsing of the incoming file is not in the scope of this tutorial. However, the parsing code has been included for completeness in the following sections.

Step 5 The Get_Token function

The get_token parses the file fetching basic tokens. Here is the function from engine.c:

The GetVars function

```
/**
 *
 * #define MAX_TOKEN_LEN 5000
 * static char _token[MAX_TOKEN_LEN]; /* Global buffer for tokens. */
 *
 **
 * Get the next token.
 *
 * @param f
 *     Open file handle. The file must be open for binary reading.
 * @return
 *     TRUE if more a token was fetched, FALSE otherwise.
 */
static Boolean_t get_token(FILE *f)
{
    int index = 0;
    char c;
    Boolean_t StopRightQuote;

    /* Skip white space. */
    while (fread(&c,sizeof(char),1,f) == 1 &&
           (c == ' ' || c == ',' || c == '\t' || c == '\n' || c == '\r'))
    {
        /* Keep going. */
    }

    if (!feof(f))
    {
        /* Now we're sitting on a non-white space character. */
        StopRightQuote = (c == '"');
        if (StopRightQuote)
        {
            _token[index++] = c;
            fread(&c,sizeof(char),1,f);
        }

        do
        {
            if (index == MAX_TOKEN_LEN-1)
                break; /* Lines shouldn't be longer than 5,000 characters. */

            if (feof(f))
                break;

            if (StopRightQuote)
            {
                if (c == '"')
                {
                    _token[index++] = c;
                    break;
                }
            }
            else
            {
                /* Note that a space or comma may terminate the token. */
                if (c == ' ' || c == ',' || c == '\t' || c == '\n' || c ==
                    '\r')
                    break;
            }
        }
    }
}
```

```

    }

    _token[index++] = c;
    fread(&c,sizeof(char),1,f);
} while(1);
}

_token[index] = '\0';

return (strlen(_token)) > 0;
}

```

This function reads a line of comma- or space-separated variables from the top of the file to be imported. The variables may optionally be enclosed in double quotes.

Converter is now complete. Recompile and load it into Tecplot 360.

```

/**
 */
static fpos_t _DataStartPos;

/**
 * Reads a line of comma or space separated variables from the
 * top of the file to be imported. The variables may optionally
 * be enclosed in double quotes.
 *
 * @param f
 *      Open file handle. The file must be open for binary reading.
 * @return
 *      TRUE if more a token was fetched, FALSE otherwise.
 */
static void GetVars(FILE *f,
                    StringList_ptr sl)
{
    char c;
    char buffer[5000];
    char *Line = buffer;
    char Var[100];
    int Index = 0;
    char Delimiter = ' ';

    /* Read up to the first new line. */
    do
    {
        if (fread(&c,sizeof(char),1,f) < 1)
            break;

        if (c != '\r' && c != '\n' && c != '\0')
            buffer[Index++] = c;
        else
            break;
    } while (1);

    buffer[Index] = '\0';

    /* Now get the variable names. */
    while (*Line)
    {
        Index = 0;
        if (*Line == '"')
        {
            /* Skip to next double quote. */
            Line++;
            while (*Line && *Line != '"')

```

```

        Var[Index++] = *Line++;
    }
    else
    {
        /* Read to the next delimiter. */
        while (*Line && *Line != Delimiter)
            Var[Index++] = *Line++;
    }

    Var[Index] = '\0';
    TecUtilStringListAppendString(sl,Var);

    /* Skip to the next non-delimiter char. */
    while (*Line && *Line != Delimiter)
        Line++;

    fgetpos(f,&_DataStartPos);

    /* Skip to next non-delimiter char. */
    while (*Line && (*Line == Delimiter || *Line == ' '))
        Line++;
}
}

```


Animating

The add-on you will create in this chapter animates the I-planes of a selected set of zones. It will appear in Tecplot 360's **Tools** menu as "Animate I Planes". *AnimIPanes* will verify that the data is IJK-ordered, change the Volume mode to I-planes, and cycle through the I-planes.

The code in this tutorial is platform-independent. All of the example source code shown in this manual is included in the Tecplot 360 distribution and is found in the *adk/samples/animiplanes* subdirectory below the Tecplot 360 home directory.



Please read [Chapter 2: "Creating Add-ons on Linux/Macintosh Platforms"](#) and/or [Chapter 3: "Creating Add-ons on Windows Platforms"](#) before continuing with this tutorial.

30 - 1 Getting Started

AnimIPanes uses source code files created by the CreateNewAddOn script (Linux/Macintosh). Our project name will be "AnimIPanes" and the add-on name will be "Animate I Planes". When running CreateNewAddOn, answer the questions as follows:

- **Project Name (Base name)** - AnimIPanes
- **Add-on name** - Animate I Planes
- **Company name** - [Your company name]
- **Type of add-on** - General Purpose
- **Language** - C
- **Use TGB to create a platform-independent GUI?** - Yes
- **Add a menu call back to the Tecplot "Tools" menu?** - Yes
- **Menu text** - Animate I Planes
- **Menu callback option** - Launch a modeless dialog
- **Dialog title** - Animate I Planes

After running the CreateNewAddOn script, you should have the following files:

```
guibld.c          guicb.c guidefs.cmain.c
```

You will also have other files specific to your platform, but we will only modify these listed files. The purpose of each file will be explained in detail as we proceed.

30 - 2 Creating the Dialog

Create the main dialog, which will be launched when *Animate I Planes* is selected from Tecplot 360's **Tools** menu. The dialog will have two labels, one button, one text field, and a multi-selection list. You will be able to select a specific set of zones to animate from the list, specify a skip level in the text field, and selecting the button will perform the animation.

1. Because we are using [Tecplot GUI Builder](#) (TGB), the dialog template is the Tecplot 360 layout file *gui.lay* (located in your project directory). Load *gui.lay* into Tecplot 360, select Tecplot GUI Builder from the **Tools** menu.



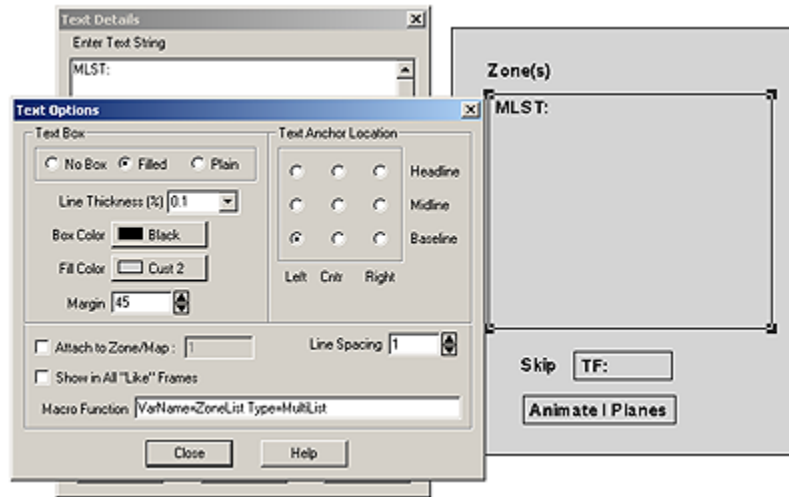
If **Tecplot GUI Builder** is not available from the **Tools** menu in Tecplot 360, refer to [Section 25 - 1 "Using Tecplot GUI Builder"](#) for instructions are running Tecplot 360 with the GUI Builder.

2. Edit the layout as follows:

You can edit a control by double-clicking on it and making the desired changes via the Text Details dialog.

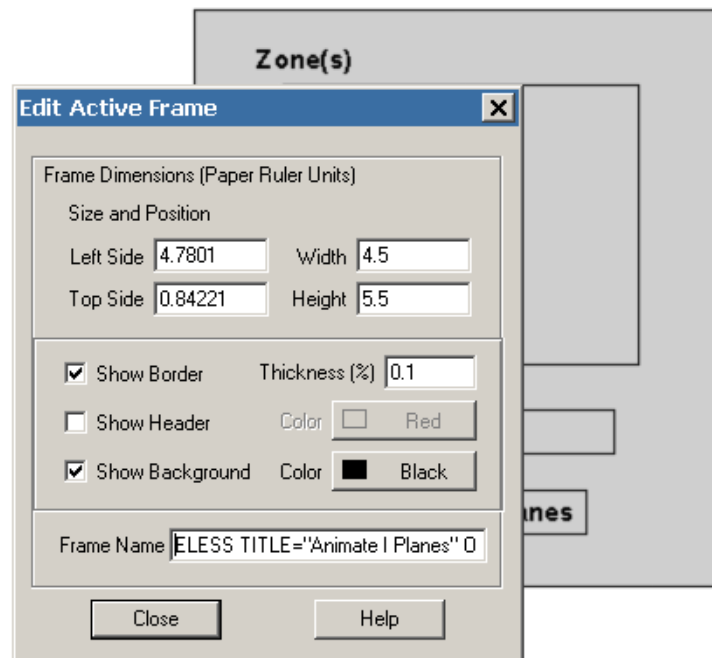
3. There will be callbacks associated with each of the text fields, and the toggle button. By default, the TGB adds generic variable names for these controls. Change these variable names by double-clicking on the control and selecting the [Options] button in the resulting **Text Details** dialog. The variable name can be changed in the Macro Function field of the resulting **Text Options** dialog.

- a. Double-click on the MLST: multi-selection list and select the [Options] button. In the Macro Function field on the **Text Options** dialog, set VarName=ZoneList. This will be the base name of the callback associated with the multi-selection list.



- b. Change the Macro Function for the TF: text field to VarName=Skip.
- c. Change the Macro Function for the Animate I Planes button to VarName=AnimPlanes.
4. The dialog title is specified in the **Edit Active Frame** dialog. Double-click on the dialog frame and verify that the frame is:

ID=1 MODE=MODELESS TITLE="Animate I Planes"



5. You can now build the source for this layout. Select Go Build from the **GUI Builder** dialog.
6. Now that TGB has created new stub files, be sure to copy the toggle and text field callbacks from *guibc.tmp* into *guibc.c*.

30 - 3 Setting Up State Variables/Initializing Dialog Fields

When the dialog is launched we need to make sure that the Skip and ZoneList text fields are filled in properly. To initialize Skip we will define the skip to be a reasonable default value and set it every time the dialog is launched. This initialization will take place in the Dialog1Init_CB function. This function is called every time the dialog is launched.

Note the following line in *guicb.c*, immediately below the #include statements:

and the following code used as the dialog initialization callback:

```
#define DEFAULT_SKIP "1"
```

```
static void Dialog1Init_CB(void)
{
    TecUtilLockStart(AddOnID);
    /*<<< Add init code (if necessary) here>>>*/
    TecGUITextFieldSetString(Skip_TF_D1, DEFAULT_SKIP);
```

```
    TecUtilLockFinish(AddOnID);
}
```

To initialize ZoneList we will write a separate function, then call that function from the Dialog1Init_CB function. This function will be called elsewhere in this exercise.

The following code is above the InitTecAddOn function:

```
void FillZoneList(void)
{
    if (TecUtilDataSetIsAvailable())
    {
        EntIndex_t NumZones, i;

        TecUtilDataSetGetInfo(NULL, &NumZones, NULL);
        TecGUIListDeleteAllItems(ZoneList_MLST_D1);
        for (i = 1; i <= NumZones; i++)
        {
            char *ZoneName;
            TecUtilZoneGetName(i, &ZoneName);
            TecGUIListAppendItem(ZoneList_MLST_D1, ZoneName);
            TecUtilStringDealloc(&ZoneName);
        }
    }
    else
        TecGUIListDeleteAllItems(ZoneList_MLST_D1);
}
```

This function will fill the zone list with the zone names of the data set in the active frame. If there is no data set, the items in the list are deleted.

This function is called in the dialog initialization callback in *guicb.c*. The callback should now look like:

```
static void Dialog1Init_CB(void)
{
    TecUtilLockStart(AddOnID);
    /*<<< Add init code (if necessary) here>>>*/
    TecGUITextFieldSetString(Skip_TF_D1, DEFAULT_SKIP);
```

```
    FillZoneList();
```

```
TecUtilLockFinish(AddOnID);
}
```

Since the function body of `FillZoneList` is in `main.c`, add the following line to `ADDGLBL.h`:

```
EXTERN void FillZoneList(void);
```

30 - 4 The Animate I Planes Button

When the Animate I Planes button is selected, we want to animate the I-planes. We will create a function called `AnimatePlanes`, and add a call to that function in the `AnimatePlanes_BTN_D1_CB` callback function.

Before calling the `AnimatePlanes` function we need to collect data from the dialog and check to see that there is a data set available. The `AnimatePlanes` function will take two parameters, `ZoneSet` and `Skip`. `ZoneSet` will contain the zones that were selected in the dialog, and `Skip` will be the skip value that was entered in the text field:

```
static void AnimPlanes_BTN_D1_CB(void)
{
    TecUtilLockStart(AddOnID);

    /* Make sure there is a dataset */
    if (TecUtilDataSetIsAvailable())
    {
        LgIndex_t Count      = 0;
        LgIndex_t *Selection = NULL;
        Set_pa      ZoneSet  = TecUtilSetAlloc(TRUE);

        /* Get the Skip value from the text field */
        char *strSkip = TecGUITextFieldGetString(Skip_TF_D1);

        /* Get the selected zones from the ZoneList */
        TecGUIListGetSelectedItems(ZoneList_MLST_D1, &Selection, &Count);
        if (Count > 0)
        {
            LgIndex_t i;

            /* Put the selected items into ZoneSet */
            for (i = 0; i < Count; i++)
                TecUtilSetAddMember(ZoneSet, Selection[i], TRUE);

            TecUtilArrayDealloc((void **)&Selection);
        }

        /* Make sure a zone has been picked */
        if (ZoneSet != NULL) /* ...do the animation */
            AnimatePlanes(ZoneSet, atoi(strSkip));
        else
            TecUtilDialogErrMsg("No zones have been picked.");

        /* Deallocate the ZoneSet and strSkip string when we are done with
        them */
        if (ZoneSet != NULL)
            TecUtilSetDealloc(&ZoneSet);
        if (strSkip != NULL)
            TecUtilStringDealloc(&strSkip);
    }
    else
        TecUtilDialogErrMsg("No data set available.");

    TecUtilLockFinish(AddOnID);
}
```

We collect the information from the dialog and then pass that information off to `AnimatePlanes` to carry out the animation. Because `ZoneSet` is initialized to `NULL`, we can tell if there were any selections. If there were not, we display an error message reading “No zones have been picked.”

30 - 5 Writing the `AnimatePlanes` Function

This function will perform the actual animation. It takes two parameters, `ZoneSet` and `Skip`. These parameters are collected in the `AnimatePlanes` button callback function in `main.c`:

```
void AnimatePlanes(Set_pa ZoneSet,
                  int Skip)
{
    LgIndex_t    MaxIndex = 0;
    EntIndex_t    CurZone;
    SetIndex_t    NumberOfZonesInSet;
    SetIndex_t    Index;
    Set_pa        IJKZoneSet = TecUtilSetAlloc(TRUE);
    char          *strMacroCommand;

    /* Get the number of zones in ZoneSet */
    NumberOfZonesInSet = TecUtilSetGetMemberCount(ZoneSet);

    if (TecUtilMacroIsRecordingActive() &&
        (NumberOfZonesInSet >= 1))
    {
        strMacroCommand = TecUtilStringAlloc(2000, "Macro Command");
        strcpy(strMacroCommand, "ZONESET=");
    }

    /*
     * Create a subset of ZoneSet that includes only
     * IJK Ordered Zones. Do this by looping through
     * all the zones in ZoneSet, check to see if the zone
     * is IJK Ordered. Then add the zone to IJKZoneSet
     */
    for (Index = 1; Index <= NumberOfZonesInSet; Index++)
    {
        /* Get the current zone */
        CurZone = (EntIndex_t)TecUtilSetGetMember(ZoneSet, Index);

        /* Make sure the current zone is enabled */
        if (TecUtilZoneIsEnabled(CurZone))
        {
            /* Only add the zone if it is IJK ordered */
            if (ZoneIsIJKOrdered(CurZone))
            {
                TecUtilSetAddMember(IJKZoneSet, CurZone, TRUE);
                /* Find the greatest IMax of all the valid IJK ordered
zones */
                MaxIndex = MAX(MaxIndex, GetIMaxFromCurZone(CurZone));
            }

            if (TecUtilMacroIsRecordingActive())
            {
                sprintf(&strMacroCommand[strlen(strMacroCommand)], "%d",
CurZone);
                if (Index != NumberOfZonesInSet)
                    strcat(strMacroCommand, ",");
            }
        }
    }
}
```

```

/* Only proceed if there is at least one IJK ordered zone */
if (TecUtilSetGetMemberCount(IJKZoneSet) >= 1)
{
    Boolean_t IsOk = TRUE;

    /* Setup the zones for animation of I-Planes */

    /* Change the cell type to planes */
    TecUtilZoneSetVolumeMode(SV_CELLTYPE,
                            NULL,
                            IJKZoneSet,
                            (ArbParam_t)IJKCellType_Planes);

    /* Display only the I-Planes */
    TecUtilZoneSetVolumeMode(SV_PLANES,
                            NULL,
                            IJKZoneSet,
                            (ArbParam_t)Planes_I);

    /* Make sure that the Skip is greater than or equal to one. */
    if (Skip < 1)
        Skip = 1;

    /* Do the actual animation */
    TecUtilDoubleBuffer(DoubleBufferAction_On);
    for (Index = 1; IsOk && Index <=MaxIndex; Index += Skip)
    {
        /*
         * Set the range of the I-Planes so that the
         * minimum I-Plane to display is the same as
         * the maximum displayed. Then increment
         * by Skip. This will make the I-Planes "move"
         */
        TecUtilZoneSetVolumeMode(SV_IRANGE,
                                SV_MIN,
                                IJKZoneSet,
                                (ArbParam_t)Index);
        TecUtilZoneSetVolumeMode(SV_IRANGE,
                                SV_MAX,
                                IJKZoneSet,
                                (ArbParam_t)Index);
        IsOk = TecUtilRedraw(TRUE);
        TecUtilDoubleBuffer(DoubleBufferAction_Swap);
    }
    TecUtilDoubleBuffer(DoubleBufferAction_Off);

    if (IsOk && TecUtilMacroIsRecordingActive())
    {
        /* At this point we have all the IJK ordered zones.
         * So all we need to add is the skip value. Add a semi-colon
         * to the end to signify the end of the IJKZoneSet information.
         */
        strcat(strMacroCommand, "; ");
        sprintf(&strMacroCommand[strlen(strMacroCommand)], "SKIP=%d",
Skip);
        strMacroCommand[strlen(strMacroCommand)] = '\0';

        /* Record the command */
        TecUtilMacroRecordExtCommand("animiplanes", strMacroCommand);
        TecUtilStringDealloc(&strMacroCommand);
    }
}

```

```

    TecUtilSetDealloc(&IJKZoneSet);
}

```

Note the use of double buffering when we do the animation. If we do not double buffer, there will be a significant amount of flickering during animation. This is due to the time it takes to draw the other zones. There are a few functions called in the previous example that have not yet been defined; they check to see if the zone passed is IJK-ordered.

Note the following functions above the `AnimatePlanes` function:

```

static Boolean_t ZoneIsIJKOrdered(EntIndex_t ZoneNum)
{
    Boolean_t IsOk;
    LgIndex_t IMax, JMax, KMax;

    TecUtilZoneGetInfo(ZoneNum,
                        &IMax,
                        &JMax,
                        &KMax,
                        NULL, /* XVar */
                        NULL, /* YVar */
                        NULL, /* ZVar */
                        NULL, /* NMap */
                        NULL, /* UVar */
                        NULL, /* VVar */
                        NULL, /* WVar */
                        NULL, /* BVar */
                        NULL, /* CVar */
                        NULL); /* SVar */

    IsOk = (IMax > 1 && JMax > 1 && KMax > 1);
    return IsOk;
}

```

This function is added for convenience, so as to not clutter `AnimatePlanes`.

```

static LgIndex_t GetIMaxFromCurZone(EntIndex_t ZoneNum)
{
    LgIndex_t IMax;
    TecUtilZoneGetInfo(ZoneNum,
                        &IMax,
                        NULL, /* JMax */
                        NULL, /* KMax */
                        NULL, /* XVar */
                        NULL, /* YVar */
                        NULL, /* ZVar */
                        NULL, /* NMap */
                        NULL, /* UVar */
                        NULL, /* VVar */
                        NULL, /* WVar */
                        NULL, /* BVar */
                        NULL, /* CVar */
                        NULL); /* SVar */

    return IMax;
}

```

Compile the add-on and make sure that it runs properly. If you have two frames with different data sets, the zone list will not be updated when switching between frames.

30 - 6 Monitoring State Changes

Now we will add functionality to allow the zone list to update properly. To do this we will need to listen for state changes. When something in Tecplot 360 changes, such as a new top frame, Tecplot 360 broadcasts a message saying that there is a new top frame. We are going to add code to our add-on to

allow it to listen for these messages. This is called a State Change Callback function. Refer to [Chapter 6: "State Changes From an Add-on"](#) for detailed information on state changes.

During the setup of this add-on we requested to have state change monitoring code included in the initial build. This code was added to *main.c*. Now locate the function `AnimIPanesStateChangeCallback` in *main.c*. Notice that it already contains a switch statement with all the state changes you can monitor. The cases that the add-on is concerned about are grouped together in the state change callback:

```
case StateChange_NewTopFrame :
case StateChange_ZonesAdded :
case StateChange_ZonesDeleted :
case StateChange_FrameDeleted :
case StateChange_ZoneName :
case StateChange_DataSetReset :
```

A call to `FillZoneList` must be performed when these state changes are detected. The resulting code should look as follows:

```
void STDCALL AnimIPanesStateChangeMonitor(StateChange_e StateChange,
                                         ArbParam_t   CallData)
{
    TecUtilLockStart(AddOnID);
    switch (StateChange)
    {
```

```
case StateChange_NewTopFrame :
case StateChange_ZonesAdded :
case StateChange_ZonesDeleted :
case StateChange_FrameDeleted :
case StateChange_ZoneName :
case StateChange_DataSetReset :
```

```
    {
        /*
         * State changes may come in here while the dialog
         * is down. We only want to fill the zone list
         * while the dialog is up.
         */
        if (TecGUIDialogIsUp(Dialog1Manager))
            FillZoneList();

        } break;
    default: break;
    }
    TecUtilLockFinish(AddOnID);
}
```

AnimIPanes is now complete. Recompile and load into Tecplot 360.

30 - 7 Exercises

1. Currently there is nothing to inform users they have entered an invalid number for the skip, such as a negative number or zero. Add error checking in the text field callback to check for a valid positive integer.
2. Check that the integer in the text field is less than or equal to the maximum I-Max for the selected zones.
3. Allow the animation of J- and K-planes. Adding an option menu to the interface with the types of planes as options would be a good place to start.

4. Add code to make the add-on remember the last skip value entered, such that when the dialog is closed and reopened the last skip value is the default in the text field.
5. Allow input of start and end planes. This would allow animation from a larger plane index to a smaller index, and allow a specific range of planes to animate.

Building add-ons with FORTRAN

To use FORTRAN on Windows machines, you must use Intel Visual FORTRAN. For the most part, only the standard FORTRAN compilers supplied with Linux platforms are supported. Other FORTRAN compilers will probably work, but you may have to customize the settings/build scripts.

A - 1 Fortran Include Files

You must at a minimum include the file FGLUE.INC in the header of each function or subroutine that calls TecUtil functions. FGLUE.INC sets the return type for TecUtil functions and also defines a number of PARAMETER values that are handy to use when calling TecUtil functions. If you are using the Tecplot GUI Builder then you should also include ADDGLBL.INC, GUIDefs.INC, and GUI.INC. FGLUE.INC and GUI.INC can be found in the include subdirectory below the Tecplot 360 home directory. ADDGLBL.INC and GUIDefs.INC are created uniquely for each add-on by the CreateNewAddOn shell script and the Tecplot GUI Builder. Thus, typical subroutines for FORTRAN add-ons to Tecplot 360 should look like:

```
SUBROUTINE MYSUB
  INCLUDE 'ADDGLBL.INC'
  INCLUDE 'FGLUE.INC'
  INCLUDE 'GUIDefs.INC'
  INCLUDE 'GUI.INC'

  C.... Code for this add-on

  RETURN
END
```

A - 2 FORTRAN Glue Functions

The FORTRAN versions of the ADK routines are documented in the [ADK Reference Manual](#) along with their C counterparts (look in Tecplot 360's doc directory). They are contained in a separate library, fglue.lib, which resides in Tecplot 360's bin (Windows) or lib (Linux/Macintosh) directory. You must link fglue.lib in with your FORTRAN add-on. A small number of C routines have no FORTRAN equivalent. These are generally related to memory allocation and deallocation not necessary with the FORTRAN API's.

A - 3 Language Calling Conventions

The FORTRAN function signatures of many TecUtil functions vary little from their C counterparts. However, due to language differences, some argument changes require additional explanation. Those differences are illustrated in this section.

A - 3.1 Sending String Parameters to Tecplot 360

Character strings used as parameters to TecUtil functions must be terminated with a character of value zero (not a "0"). This is necessary because the TecUtil functions really call C glue functions, and strings must be terminated with a 0 character value in C.

For example, calling the function TecUtilDialogMessageBox with the string "Hi Mom" will look like:

```
I = TecUtilDialogMessageBox('Hi Mom'//char(0),  
& MESSAGEBOX_INFORMATION)
```

A - 3.2 Receiving String Parameters from Tecplot 360

In order to take advantage of FORTRAN's native strings the TecUtilStringAlloc and TecUtilStringDealloc functions are not provided. As a result, all TecUtil functions in the C layer that either returned an allocated string as a function return value or did so by modification of an output parameter have been changed to use FORTRAN native strings. In addition, an accompanying length argument is also included so you can determine the actual length of the string.

Example:

Use the function TecUtilDialogGetSimpleText to prompt the user for their name.

```
INTEGER*4 IErr  
INTEGER*4 NameLen  
CHARACTER*80 UserName  
IErr = TecUtilDialogGetSimpleText('Enter your name'//char(0),  
& 'Joe Blow'//char(0),  
& UserName  
& NameLen)  
Write(*,*) 'Name = ',UserName(1:NameLen),'<- '
```

In the preceding example, the variable UserName is filled in with the text of the users name. NameLen then is used to tell you how many of the 80 characters available in UserName were used. The resulting write statement will have the '<-' placed after the users name.

A - 3.3 Handle Parameters

Many function in the TecUtil C layer allocate objects that are passed back and forth to the TecUtil layer, but are never manipulated directly by the add-on. String lists and sets are two examples of these objects, and they may only be manipulated via a collection of TecUtil functions. Objects of this nature are referenced in FORTRAN by the POINTER notation.

For example, suppose an add-on defined a subroutine to save variables 1 and 3 of zone 5 of the current data set. Two of several arguments required by the TecUtilWriteDataSet function are a zone and variable set. The TecUtil layer provides several functions for manipulating Tecplot 360 sets. The code to perform this operation might appear as follows (error handling removed for simplicity):

```
SUBROUTINE WriteMyData(FileName),  
CHARACTER*(*) FileName  
INCLUDE 'FGLUE.INC'  
POINTER (ZoneSetPtr, ZoneSet)  
POINTER (VarSetPtr, VarSet)  
CHARACTER*256 FileNameZ
```

```

C
c...allocate and populate the zone set: [5]
C
TecUtilSetAlloc(ZoneSetPtr, TRUE)
TecUtilSetAddMember(ZoneSetPtr, 5, TRUE)
C
c...allocate and populate the variable set: [1,3]
C
    TecUtilSetAlloc(TRUE, VarSetPtr)
    TecUtilSetAddMember(VarSetPtr, 1, TRUE)
    TecUtilSetAddMember(VarSetPtr, 3, TRUE)
C
c...be sure to pass a zero terminated string to TecUtil layer
C
    FileNameZ = TRIM(FileName)//char(0)
C
c...write the dataset to the specified file name
C
IsOk = TecUtilWriteDataSet(FileNameZ, TRUE, TRUE, TRUE, TRUE,
    & ZoneSetPtr, VarSetPtr, TRUE, TRUE, TRUE)
C
c...cleanup allocations
C
    TecUtilSetDealloc(ZoneSetPtr)
    TecUtilSetDealloc(VarSetPtr)
END

```

The POINTER variables, ZoneSetPtr and VarSetPtr, are the handles used to access the respective sets and are passed to the TecUtilWriteDataSet function while the POINTEE variables, ZoneSet and VarSet, are dummy variables and should not be used.

Two exceptions to this rule are TecUtilDataNodeGetRawPtr and TecUtilDataValueGetRawPtr, where the POINTEE variables are arrays to Tecplot 360's internal data arrays and may be manipulated directly.

A - 4 Special Parameter Values

The include file FGLUE.INC mentioned in [Section A - 1 "Fortran Include Files"](#) not only declares the return types for all TecUtil functions, it also defines a large number of PARAMETER values useful for inclusion in parameters to TecUtil functions. In the example of the previous section, the second parameter in the call to TecUtilDialogMessageBox is MESSAGEBOX_INFORMATION. If you look in the FGLUE.INC file (located in the include directory below the Tecplot 360 home directory) you will find:

```

INTEGER*4 MESSAGEBOX_INFORMATION
PARAMETER(MESSAGEBOX_INFORMATION = 2)

```

It is best to always use the PARAMETER name instead of the number. It makes your code more readable, prevents errors, and makes it easier to upgrade later if the underlying value changes. In the [ADK Reference Manual](#), you can see what the possible values for a parameter are by looking at the C equivalent of the TecUtil function. In C, these PARAMETER values are defined as "enumerated types" and are listed with the description of the parameter. In C you must use the correct case when typing in the parameter name, but this is not necessary in FORTRAN.

A - 5 Checking FORTRAN Source Using the fcheck Utility

Provided in the distribution is a Linux shell script called fcheck. Use fcheck to examine FORTRAN source files. Fcheck will report any misuse of functions (such as calling a sub-routine as if it were a function) and will warn you about all functions that have changed and or been removed.

To use fcheck type:

```
fcheck fileList
```

where *filelist* is one or more FORTRAN source files (*.F).

A - 6 Issues on Windows Platforms

The following issues are specific to FORTRAN add-on development on Windows operating systems.

A - 6.1 Calling Conventions

Intel Visual FORTRAN's default calling convention is `__stdcall`, with routine names all upper-case. All parameters are passed by reference, and the length of string parameters is passed immediately after the strings themselves. This is in contrast to Linux, where string length parameters are appended to the list of parameters. It is important to use these default settings when you build your add-on, since library `fglue.lib` assumes that you are using them.

A - 6.2 Compiler Directives

Intel Visual FORTRAN must be instructed to export your add-on's `InitTecAddOn113` routine so that Tecplot 360 can call this routine when it loads your add-on. The following directive performs this action:

```
!DEC$ attributes DLLEXPORT::InitTecAddOn113
```

GUI-related routines called by Tecplot 360 when the user interacts with your add-on's interface must use the `__cdecl` calling convention. Since, as noted previously, this is not the default, the compiler must be directed to use this convention using the following directive:

```
!DEC$ attributes C::<subroutine name>
```

The Tecplot GUI Builder's generated source includes this directive where needed.

Refer to your compiler's documentation for more information on these compiler directives.

A - 6.3 Writing to the Console

On Windows operating systems, Tecplot 360 add-ons do not have access to a console by default. Therefore, statements intended to write to the console (e.g. `write(*,*)`) will not have their intended effect. In fact, Visual FORTRAN Version 6 actually causes Tecplot 360 to quit when such a statement is encountered. For this reason, you must either avoid writing to the console, or open a console to receive the output. If you wish to open a console, place the following code in `main.F`:

```
subroutine ShowConsole
  USE DFWIN
  logical status
  C Get a console window of the currently set size:
  status = AllocConsole!
  return
end
subroutine HideConsole
  USE DFWIN
  logical status
  status = FreeConsole
  return
end
```

Call `ShowConsole` from your `InitTecAddOn113` routine. `HideConsole` is probably optional, but can be called from `StateChangeCallback` for the state change `StateChange_QuitTecplot`.

Alternatives to console output include writing to a file instead, or calling `TecUtilDialogMessageBox` or `TecUtilDialogErrMsg` to report information to the user.

Part 8 Appendixes

Migrating Add-ons

B - 1 Migrating Add-ons

B - 1.1 Active Frame Manipulation

The method of specifying and manipulating the active frame has changed in Tecplot 360 2009. Because of this addition, the following TecUtil functions have been deprecated:

Deprecated Function	Use Instead
TecUtilFramePopByName	TecUtilFrameMoveToTopByName and TecUtilFrameActivateTop
TecUtilFramePushByName	TecUtilFrameMoveToBottomByName and TecUtilFrameActivateTop
TecUtilFramePopByUniqueID	TecUtilFrameMoveToTopByUniqueID and TecUtilFrameActivateTop
TecUtilFramePushByUniqueID	TecUtilFrameMoveToBottomByUniqueID and TecUtilFrameActivateTop
TecUtilFramePushTop	TecUtilFrameMoveToBottomByNumber and TecUtilFrameActivateTop
TecUtilFramePush	TecUtilFrameMoveToBottomByNumber and TecUtilFrameActivateTop
TecUtilFramePop	TecUtilFrameMoveToTopByNumber and TecUtilFrameActivateTop
TecUtilFramePopAtPosition	TecUtilFramePopAtPosition and TecUtilFrameActivateTop
TecUtilFrameDeleteTop	TecUtilFrameDeleteActive
TecUtilFrameLightweightPopStart	TecUtilFrameLightweightLoopStart
TecUtilFrameLightweightPopNext	TecUtilFrameLightweightLoopNext
TecUtilFrameLightweightPopEnd	TecUtilFrameLightweightLoopEnd
TecUtilPickAtPosition	TecUtilPickAddAtPosition

A few state changes have been deprecated as well:

Deprecated State Changes	Use Instead
StateChange_NewTopFrame	StateChange_NewActiveFrame and/or StateChange_FrameOrderChanged

Refer to the [Tecplot 360 Scripting Guide](#) and the [ADK Reference Manual](#) for details on working with the new frame activation capabilities.

B - 1.2 Surface Clipping TecUtil Changes

With the addition of surface clipping abilities in Tecplot 360 2009, the TecUtil functions

TecUtilStyleSetLowLevelX (ArgList_pa Arg List), and

TecUtilStyleGetLowLevelX(Arg_List_pa ArgList)

received new arguments:

Attribute	Added Argument	Definition
SV_SLICEATTRIBUTES	MS_OBEYCLIPPING	Specifies whether the slice should obey clipping planes
SV_SLICEATTRIBUTES	MS_CLIPBELOWSLICE	Turns on clipping plane from min value on axis towards the max value of slice
SV_SLICEATTRIBUTES	MS_CLIPABOVSlice	Clips from the max value on the axis towards the min value of slice
SV_ISOSURFACEATTRIBUTES	MS_OBEYCLIPPING	Specifies whether the isosurface is clipped by clipping planes
SV_STREAMATTRIBUTES	MS_OBEYCLIPPING	Specifies whether streamtraces are clipped by clipping planes
SV_FIELDMAP: MS_EFFECTS	MS_MS_CLIPBY	Specifies a set of clipping planes which the zone observes

B - 1.3 Export Format TecUtil Changes

With the addition of the Tecplot Viewer export format in Tecplot 360 2009, the TecUtil function TecUtilExportSetup has gained a new enumerator of ExportFormat_TecplotViewer as one of the ExportFormat_e enumerations.

B - 1.4 Load-on-demand for Connectivity Lists and Face Neighbors

Load-on-demand for connectivity lists and face neighbors has been added to Tecplot 360 2008 Release 2 and newer versions. Because of this addition, the following TecUtil functions have been deprecated:

```

TecUtilDataNodeGetRawPtr
TecUtilDataNodeGetRef
TecUtilDataFaceNbrArrayAssign
TecUtilDataFaceNbrAssign
TecUtilDataFaceNbrBeginAssign
TecUtilDataFaceNbrBeginAssignX
TecUtilDataFaceNbrEndAssign
TecUtilDataFaceNbrGetByRef
TecUtilDataFaceNbrGetByZone
TecUtilDataFaceNbrGetRawPtr
TecUtilDataFaceNbrGetRef

```

Refer to [Section 27 - 7 "Loading the Connectivity List"](#) and [Section 27 - 8 "Loading Face Neighbor Data"](#) for details on working with the new loading capabilities.

B - 2 Migrating Add-ons from Tecplot 360 2006 or Earlier to Tecplot 360 2008 R2 or Later

Tecplot 360 2008 introduced many new features and performance enhancements to benefit the add-on developer. Add-ons, data loaders specifically, can take advantage of new polyhedral and polygonal zone types, shared grid file types, Python scripting and a multitude of new TecUtil functions.

B - 2.1 Polyhedral Support

Tecplot 360 2008 and newer versions support polyhedral and polygonal zone types. These are specified in an add-on using `ZoneType_FEPolyhedron` and `ZoneType_FEPolygon`. Polyhedral and polygonal data consists of blocks of data that define nodes for faces and faces for cells. Polyhedral face maps specify the number of nodes for each face, the nodes for each face and the left and right neighbors of each face. Polygonal face maps always have two nodes per face so only the nodes for each face and the left and right neighbors of each face need to be specified. If a face has multiple neighbors on a side or one or more of its neighbors belong to another zone, the left or right element value points to an offset into the boundary connections of the face map. See the section on [Section 3 - 8 "Defining Polyhedral and Polygonal Data"](#) in the [Data Format Guide](#) for more details on working with face maps.

The complexity of these new zone types requires the use of new TecUtil functions including `TecUtilDataFaceMapCustomLOD`, `TecUtilDataFaceMapBeginAssign`, `TecUtilDataFaceMapEndAssign`, `TecUtilDataFaceMapAssignNodes` and `TecUtilDataFaceMapAssignElems`. These, and other, new functions are described following and are detailed in the [ADK Reference Manual](#).

The following functions were added for polyhedral support:

- `TecUtilDataFaceMapAlloc` - allocates space for the face map of a zone.
- `TecUtilDataFaceMapCustomLOD` - registers the load-on-demand callbacks to load face map data into Tecplot 360. Loading face maps on demand allows for better performance, both in time and space.
- `TecUtilDataFaceMapGetClientData` - gets the client data to use in a face map load-on-demand callback.
- `TecUtilDataFaceMapGetReadableRef` - gets a readable reference to the face map
- `TecUtilDataFaceMapGetWritableRef` - gets a writable reference to the face map.
- `TecUtilDataFaceMapBeginAssign` - allows an add-on to begin assigning face map data to Tecplot for a specific zone. Upon completion, `TecUtilDataFaceMapEndAssign` is called.
- `TecUtilDataFaceMapAssignNodes` - assigns the number of nodes per face and the nodes themselves for each face in the zone. The number of nodes per face is not needed for polygonal zones (as they always have two nodes per face).
- `TecUtilDataFaceMapGetFaceNode` - retrieves a node from a face.
- `TecUtilDataFaceMapGetNFaceNodes` - retrieves the number of nodes for a face.
- `TecUtilDataFaceMapAssignElems` - assigns the left and right neighboring cells for each face in the zone.
- `TecUtilDataFaceMapGetLeftElem/TecUtilDataFaceMapGetRightElem` - retrieve the left and right neighboring cell for a face.
- `TecUtilDataFaceMapAssignBConns` - assigns any boundary connections for the zone, connecting faces with multiple neighbors or neighbors in other zones.
- `TecUtilDataFaceMapGetBndryConn` - retrieves the element and zone connected to a boundary face.
- `TecUtilDataFaceMapGetNBndryConns` - retrieves the number of boundary connections for a face.
- `TecUtilDataElemGetReadableRef` - retrieves a readable reference to the element-to-face map.
- `TecUtilDataElemGetNumFaces` - retrieves the number of faces for an element.
- `TecUtilDataElemGetFace` - retrieves the face number for a face in an element.
- `TecUtilTecPoly` - writes out the face map data in Tecplot's binary file format.



The `FaceBndryConnectionZones` parameter for `TecPoly112` has been changed from an `INTEGER2` to an `INTEGER4`.

B - 2.2 Shared Grid Files

Grids that do not change in structure can be shared across files by using Tecplot 360's shared grid format. Any data that will be shared across files can be specified in the "grid" file, while any data that changes over time is specified in a "solution" file for each time step. Shared grids can also be used for moving meshes, if the connectivity is static. Shared grid files are only read by Tecplot 360 and not written out. For more information on Tecplot's shared grid format, see the [Data Format Guide](#).

B - 2.3 Python Scripting

Tecplot 360 2013 R1 supports Python scripting. Simple add-ons can be written in Python, but it should be noted that callbacks are not supported through Python. Refer to the chapter on [Chapter 30: "Working With Python Scripts"](#) in the [User's Manual](#) for more details on using Python scripting with Tecplot 360.

B - 2.4 New in Tecplot 360 2013 R1: Functions for Active Frame Manipulation

Because Tecplot 360 2013 R1 allows the user to manipulate a frame without popping that active frame to the top, additional TecUtil functions and state changes were added.

TecUtil Functions

The following TecUtil functions were added for frame management support:

- TecUtilFrameActivateTop - Activates the top frame in the drawing order
- TecUtilFrameActivateByName - Activates a frame by name
- TecUtilFrameActivateByNumber - Activates a frame by number
- TecUtilFrameActivateByUniqueID - Activates a frame by unique ID
- TecUtilFrameActivateAtPosition - Activates a frame by position
- TecUtilFrameDeleteActive - Deletes the active frame
- TecUtilFrameLightweightLoopStart - Begins a loop through all frames without changing frame drawing order. Progresses from bottom to top of frame draw order
- TecUtilFrameLightweightLoopNext - Loops to the next frame without changing frame drawing order. Progresses from bottom to top of frame draw order
- TecUtilFrameLightweightLoopEnd - Ends loop through all framees without changing frame drawing order
- TecUtilFrameMoveToTopByName - Moves frame specified by name to top of frame draw order (front)
- TecUtilFrameMoveToTopByNumber - Moves frame specified by number to front
- TecUtilFrameMoveToTopByUniqueID - Moves frame specified by unique ID to front
- TecUtilFrameMoveToBottomByName - Moves frame specified by name to bottom of draw order (back)
- TecUtilFrameMoveToBottomByNumber - Moves frame specified by number to back
- TecUtilFrameMoveToBottomByUniqueID - Moves frame specified by unique ID to back
- TecUtilPickAddAtPosition - Allows the user to select a frame to add, and allows user to specify ConsiderStyle parameter

State Changes

The following state changes were added:

- StateChange_NewActiveFrame - Which frame is the active frame has changed

- StateChange_FrameOrderChanged - Frame drawing order has changed

B - 2.5 New in Tecplot 360 2008: TecUtil Functions

- TecUtilCreateSphericalZone - creates a spherical IJ-ordered zone.
- TecUtilCreateSliceZoneShowTrace - allows activation of an arbitrary slice identifier.
- TecUtilConvertGridTo3DPosition - converts view coordinates to world coordinates.
- TecUtilDialogGetFolderNameX - launches a dialog to get a folder name.
- TecUtilStateChangeRemoveCBX - removes a callback from Tecplot 360's list of add-on state change callbacks.
- TecUtilMenuAddStatusLineHelp - registers a string to be added to the status line for a menu item.
- TecUtilThreadCreateDetached - creates a new thread to execute a specific function.
- TecUtilScriptExecRegisterCallback - registers a script execution callback.
- TecUtilScriptExec - executes a script file.
- TecUtilScriptProcessorGetClientData - retrieves the client data for a script processor.
- TecUtilTecplotGetExePath - retrieves the full path of the Tecplot 360 executable.
- TecUtilAddOnGetPath - retrieves the full path of an add-on.
- TecUtilGetBoundingBoxOfAllFrames - gets the dimension of the bounding box surrounding all frames.
- TecUtilDataFECellGetUniqueNodes - retrieves the unique nodes for a cell or face of a volume cell.
- TecUtilDataValueGetReadableNativeRef - retrieves a readable reference to the native data of a zone variable.
- TecUtilDataValueGetReadableDerivedRef - retrieves a readable reference to the derived data of a zone variable.
- TecUtilDataValueGetReadableNLRef - retrieves a readable reference to the node located data of a zone variable.
- TecUtilDataValueGetReadableCCRef - retrieves a readable reference to the cell centered data of a zone variable.
- TecUtilDataValueGetWritableNativeRef - retrieves a writable reference to the native data of a zone variable.
- TecUtilFieldMapGetActive - retrieves the set of active fieldmaps.
- TecUtilFieldMapIsActive - queries if a fieldmap is active.
- TecUtilFieldMapHasOrderedZones - queries if a fieldmap contains any ordered zones.
- TecUtilFieldMapHasIJKOrderedZones - queries if a fieldmap contains any IJK ordered zones.
- TecUtilFieldMapHasFEZones - queries if a fieldmap contains any FE zones.
- TecUtilFieldMapHasVolumeZones - queries if a fieldmap contains any FE volume zones.
- TecUtilFieldMapHasSurfaceZones - queries if a fieldmap contains any FE surface zones.
- TecUtilFieldMapHasLinearZones - queries if a fieldmap contains any linear zones.
- TecUtilDataValueIsPassive - queries if a zone variable is passive.
- TecUtilGetDefaultExportImageWidth - calculates the width of an image for exporting.

B - 3 Migrating Add-ons From Version 10 to Tecplot 360 2006

B - 3.1 Data Loaders

As technology advances, datasets and zones get larger and larger. With the introduction of Load-On-Demand, data loaders can now take advantage of substantial performance increases, especially with large data. The uses of Load-On-Demand are discussed in detail in [Section 27 - 5 "Load-on-demand"](#). We highly recommend that your data loaders be updated to use this powerful new feature.

B - 3.2 Time Aware

Transient data can now be loaded into Tecplot 360 on a time step basis. Each zone must have a "Strand ID" and a "Solution Time" to be "time aware". There now exists a host of new functions for add-ons. These functions are discussed in detail in the [ADK Reference Manual](#). The StrandID for a zone is set when the zone is added to the dataset but `TecUtilZoneGetStrandID` can be used to retrieve it. Solution time can also be included with a zone as it is added to the dataset, but data loaders may need to set the solution time later on. `TecUtilZoneGetSolutionTime` will get the solution time and `TecUtilZoneSetSolutionTime` will set the solution time for a specific zone. `TecUtilSolutionTimeGetCurrent` will return the current solution time while `TecUtilSolutionTimeSetCurrent` will set the current solution time for the active frame. `TecUtilZoneGetRelevant` will get the set of zones at a specific solution time.

B - 3.3 Data Functions

Read/Write Versions of Data Functions

In previous versions of the ADK, there was a single set of `TecUtil` functions to be used for both reading and writing data.

In Tecplot 360, we've added optimized versions of these functions specifically for the separate tasks of reading or writing of data. You can still use the old functions, but the new functions provide the highest speed and efficiency.

Deprecated V10 Function	Equivalent 360 Function
<code>TecUtilDataValueGetRawPtr</code>	<code>TecUtilDataValueGetReadableRawPtr</code> or <code>TecUtilDataValueGetWritableRawPtr</code>
<code>TecUtilDataValueGetRef</code>	<code>TecUtilDataValueGetReadableRef</code> or <code>TecUtilDataValueGetWritableRef</code>

Querying or Setting Variable Min/Max

You can easily query the Min and Max of a variable by calling: `TecUtilDataValueGetMinMaxByZoneVar`.

For large datasets, calculating the min and max of a variable is expensive. If you already know the min and max of your data, you can provide this information to Tecplot 360 so that Tecplot 360 will not have to calculate it. Use either of these new functions in Tecplot 360:

`TecUtilDataValueSetMinMaxByRef`

`TecUtilDataValueSetMinMaxByZoneVar`

Other TecUtil Functions New in Tecplot 360

TecUtilDialogGetFileNamesX

In the 'Filter Types' parameter, you can now supply a list of types in a single string, where each type is separated with a vertical bar.

The list is ended with two vertical bars.

Example:

```
// select all files or text files only
TecUtilArgListAppendStringList (argList,
SV_FILESELECTDIALOGFILTEREXTENSIONS, "All files (*.*)|*.*|Text
files(*.txt)|*.txt|");
Result = TecUtilDialogGetFileNamesX(argList);
```

TecUtilImportGetLoaderInstrCount

You can now query the loader instructions for the n^{th} loader. Use TecUtilImportGetLoaderInstrCount to find the number of loader instructions available

TecUtilImportGetLoaderInstrByNum

You can now query the loader instructions for the n^{th} loader. Use TecUtilImportGetLoaderInstrByNum to query the loader instructions for the n^{th} loader.

TecUtilToolbarActivate

If your add-on activates or deactivates the sidebar and/or Tecplot 360 menus, you will probably want to deactivate/activate the toolbar also. Use this function to do it.

TecUtilAnimateTimeX

Use this function instead of TecUtilAnimateZones when dealing with time aware data.

Index

Symbols

\$!ADDONCOMMAND command 199
\$!LoadAddOn command 40

A

AbbreviatedSettingsStringCallback function 300
ADDGLBL.h 80
 in SimpAvg 292
Adding a dialog 31
Adding dialogs or controls 227
Add-On Development Root Directory 23
Add-on initialization and mopup 45
-addonfile command 39
ADDONGLB.h 26, 37
Add-ons
 \$!LoadAddOn command 40
 accessing field data 75
 Animate I Planes button 319
 assigning to Quick Macro buttons 199
 Compute function writing 84
 Converter 306
 creating 24
 curve-fit add-on design 291
 data converters 245
 data loaders 245
 dialog field initialization 83
 dialog initialization 296
 dialog launch 296
 dialogs 316
 Equate 80
 Equate dialog creation 80
 exercises 85, 241, 304, 323
 Help 209
 isolating during development 40
 MenuCallback modification 26, 37
 mopup 46
 online help 209
 register in Tecplot 295
 running 39
 specifying on command line 39
 specifying which to load 39
 state change callbacks 323
 state changes 322
 state variable set up 83
 SumProbe 239
 testing 40
Add-ons loaded by all users 39
Advanced topics
 Equate exercises 85
AFX_MANAGE_STATE 31
Animate I Planes add-on
 creating dialogs 316
AnimatePlanes function 319, 320, 322
AnimatePlanes_BTN_D1_CB function 319
Animation 169
 double buffering 322
AnimIPlanes add-on
 Animate I Planes button 319
 exercises 323
Annotations 153–165
 geometry 162

 images 164
 labels 165
 text 153–155
ASCII characters 219
Auxiliary Data 93, 107–108
Axes 151–153
 grid area 153
 line plot 119
 lines 153
 range 152
 tick marks 152

B

Bars Layer 117
Blanking 165
Boundary connection 70
Boundary face 70
Building Add-Ons with FORTRAN 325
Building data set reader add-ons 245, 331
Building the source code 237
BuildSidebarNN 228

C

C compiler directive 328
Calling conventions
 __cdecl 328
 __stdcall 328
 __cdecl
 calling convention 328
Code
 examples 20, 80, 239, 279, 306, 315
Code Generator 175–179
 incorporating output 177
 launching 177
 style hierarchy 175
 undisplayed commands 178
Coloring 146–150
 basic colors 147
 multi-color 147
 rgb coloring 148
Colormap 147
Command string 200
Compiler directives 328
 C 328
 DLEXPORT 328
Compiling
 -debug 25
 -release 25
 using Runmake 25
Compiling the add-on 25
Compiling your add-on
 UNIX or Windows 238
Compute
 writing 84
Compute function 83
Connected boundary face 70
Connectivity List 63–73
 face neighbor 71
 finite-element 65
 nodal data 66
 loading 265
 classic finite-element 265
 face-based finite-element 266
 ordered data 63

INDEX

- face neighbors 64
- Console output 328
- Contour Layer 124
- Control options in TGB 229
- Controls
 - types and keywords 229
- Converter add-on
 - about 306
- ConverterCallback function 306
 - modifying 307
- Coordinate Transformation 95
- CreateCurveSettingsString function 292
- Created files
 - generated by TGB 237
- CreateNewAddOn 26, 30, 36, 37, 80
- Creating add-ons
 - Add-On Development Root Directory 23
 - creating add-ons under UNIX 23
 - creating new add-ons 24
 - setting up to build add-ons under UNIX 23
 - under Windows 29
- Creating an add-on using Visual Studio 30, 34
- Creating an Add-On with Visual C++ 30, 34
- Crossing platforms
 - converting add-ons 181
- Curve fits
 - calculating 274
 - curve setting text field 278
 - external 273
 - information 277
 - registering 273
 - settings 277
- Curve Information dialog 279
- Curve Types, line plot 120
- Curve-Fit dialog 301
- CurveInfoStringCallback function 290, 303
- CurveParams 299
- CurveParams_s 292, 293
 - description 292
- CurveSettings variable 284
 - in SimpAvg 292
- CurveSettingsCallback function 296
- CustomMake
 - editing the CustomMake file 25

D

Data

- alter 87–106
 - auxiliary data 93
 - derivative functions 91
 - difference functions 91
 - share variable 94
 - specify index 93
 - specify zone number 93
 - syntax 88
- auxiliary data 93, 107–108
- calculation 87–106
 - auxiliary data 93
 - difference functions 91
 - share variable 94
 - specify zone number 93
 - syntax 88
- code layering 251
- core loader function 252

- create zones 95–100
 - by entering values 100
- circular zone 97
- cylindrical zone 97
- duplicate zone 98
- FE surface 100
- mirror zone 99
- rectangular zone 96
- extract sub-zone 101
- face neighbor data 71
- fieldmap, create new from existing 100
- finite-element 65
 - nodal data 66
- hierarchy 248
- interpolation 101–106
 - inverse-distance 102
 - kriging 104
 - linear 101
- journaling
 - deactivation 206
 - example 206
 - prerequisites 206
- loading
 - primary steps 250–251
- metadata 107–108
- ordered 63
 - face neighbors 64
- organization 63–73
- shared 78
- sharing 94
- smoothing 94
 - limitations 94
- structure 63–73
- transform coordinates 95
- transient data 170
- triangulation 105

Data converters 245

Data loaders

- about 245

Data Loading

- append data 269
- code layering 251
- connectivity list 265
- face neighbor data 268
- finite-element data 265–269
 - polyhedral 266
- immediate loading 252–255
- load-on-demand 255–263
 - auto-load variable-on-demand 257
 - callbacks 263
 - load value-on-demand 262
 - load variable-on-demand 260
 - threads 263
- override journal 270
- primary steps 250–251
- replace data 270

Data set reader add-ons

- building 245, 331

Data set readers

- data set loaders 306

Data Structure 63–73

- face neighbor data 71
- finite-element data 65
- nodal data 66

- ordered data 63
 - face neighbors 64
- DataaFName
 - in Converter 307
- Debug 27
- debug flag 25
- Default files
 - created by TGB 237
- DepCurveValues array 285
- Developer Studio
 - used with TGB 225
- Developing add-ons
 - isolating and testing 40
- Developing Add-Ons in UNIX 40
- Developing Add-Ons in Windows 41
- Dialog
 - adding 31
- Dialog1HelpButton function 211
- Dialog1Init_CB function 83, 298, 318
- Dialogs
 - adding or creating 227
 - creating 80
 - creation 316
 - Curve Information 279
 - Curve-Fit 301
 - fields 83
 - in SimpAvg 299
 - XY-Plot Curve Info 290
 - Zone Style 295, 300
- Dialogs in Windows
 - modal and modeless 221
- Directives, compiler 328
- Display
 - animation 169
 - axes 151–153
 - blanking 165
 - code generator 175–179
 - incorporating output 177
 - launching 177
 - undisplayed commands 178
 - coloring 146–150
 - basic colors 147
 - multi-color 147
 - rgb coloring 148
 - field plot 122–143
 - derived objects 135
 - fieldmap layers 122–143
 - fieldmap 111
 - example 114
 - styles 115
 - geometry 162
 - images 164
 - labels 165
 - line plot 115–122
 - curve types 120
 - line legends 121
 - mapping layers 117, 119
 - linemap 111
 - example 112
 - styles 115
 - plot style 111–174
 - plotting data subset 143
 - points 143
 - surfaces 144
- style hierarchy 175
- text 153–155
- transient data 170
- view options 172
 - rotation 172
 - translation 173
 - zoom 173
- DLL 29
- DLLEXPORT
 - compiler directive 328
- DoConversion function 308
 - writing 308
- Double buffering
 - in animation 322
- Dynamically linked libraries
 - loading add-ons 40
- Dynamic-link libraries 19
- E**
 - Edge Layer 133
 - Engine.c
 - description 306
 - in Converter 306, 311
 - engine.c
 - in PolyInt 281, 287, 290
 - in SimpAvg 292, 296, 300
 - Engine.h
 - in SimpAvg 293, 294
 - Environment variables
 - TECADDONDEVDIR 24
 - TECADDONDEVPLATFORM 24
 - Equate
 - adding help 211
 - creating dialogs 80
 - writing Compute function 84
 - Equate add-on 80
 - Exercises 85
 - equate.html 211
 - Equation, calculate 87–94
 - auxiliary data 93
 - derivative functions 91
 - share variable 94
 - syntax 88
 - Error Bars Layer 117
 - Error messages in command callbacks 200
 - Errors
 - callback function processing 200
 - Example add-on
 - MFC DLL 31
 - non-MCF DLL 33, 35
 - Examples
 - code 20, 80, 239, 279, 306, 315
 - creating Equate dialog 80
 - Equate add-on 80
 - files 20, 80, 239, 279, 306, 315
 - source code 20, 80, 239, 279, 306, 315
 - string lists 188
 - using sets 193
 - Excercises 85
 - Exercises
 - AnimlPlanes add-on 323
 - Equate add-on 85
 - extending SimpAvg add-on 304
 - extending SumProbe add-on 241

ExtractCurveValuesFromWorkingArray function 286

F

Face Neighbor Data
 data structure 71
 loading 268
 ordered data 64
 Face Neighbors
 polyhedral zones 69
 Face neighbors
 right-hand rule 69
 Facemap data
 polyhedral zones 68
 fglue.lib 325
 Field data 75
 FieldData_pa
 in Equate 84
 Fieldmap 97–100, 111–115
 create circular zone 97
 create cylindrical zone 97
 create from existing data 100
 create rectangular zone
 duplicate zone 98
 enter values to create zone 100
 extract sub-zone 101
 FE surface zone 100
 layers 122–143
 contour 124
 edge 133
 mesh 123
 scatter 131
 vector 129
 mirror zone 99
 specify index 93
 specify zone number 93
 triangulation, irregular data point 105
 File*
 about 308
 Files
 examples 20, 80, 239, 279, 306, 315
 Files created by TGB 237
 FillZoneList function 319, 323
 Finite-element Data 65–71
 data structure 65–71
 nodal data 66
 loading 265
 classic finite-element 265
 face-based finite-element 266
 FORTRAN
 building add-ons 325
 Frame Linking 166
 Functions
 AbbreviatedSettingsStringCallback 300
 about Get_Vars 312
 AnimatePlanes 319, 320, 322
 AnimatePlanes_BTN_D1_CB 319
 Compute 83
 ConverterCallback 306
 CreateCurveSettingsString 292
 CurveInfoStringCallback 290, 303
 CurveSettingsCallback 296
 Dialog1HelpButton 211
 Dialog1Init_CB 83, 298, 318
 DoConversion 308

DoConversion writing 308
 ExtraCurveValuesFromWorkingArray 286
 FillZoneList 319, 323
 get_token 308, 311
 GetVars 308
 GUI_TextFieldSetString 83
 InitializeCurveParams 294
 InitTecAddOn 26, 37, 240, 246, 307, 318
 InsertProbeValueInWorkingArray 289
 MenuCallback 26, 37, 239
 modifying ConverterCallback 307
 MyProbeCallback 240
 PolyInt 281, 284, 287
 PrepareWorkingArray 285
 SimpleAverage 280, 302, 303
 StateChangeCallback 323
 TecIO 245
 TecUtil 239, 308
 TecUtilCurveRegisterExtCrvFit 280
 TecUtilCurveSetExtendedSettings 294
 TecUtilDialogGetVariables 240
 TecUtilImportAddLoader 246
 TecUtilMenuAddOption 26, 37, 240
 TecUtilProbeInstallCallback 240
 TecUtilStringAlloc 292
 TecUtilTecDat 308
 TecUtilTecEnd 308
 TecUtilTecIni 308
 TecUtilTecZne 308
 TecUtilVarIsEnabled 85
 TecUtilZoneIsEnabled 85
 UpdateMainDialogSensitivities 298
 writing Compute 84
 XYDataPointsCallback 284, 285, 287, 301

G

Generated files
 created by TGB 237
 Geometry 162
 labels 165
 Get_token function 308, 311
 Get_Vars function
 about 312
 GetVars function 308
 Grid area 153
 GUI
 building source code 237
 control types and keywords 229
 GUI builder 25
 GUI Source Code 82
 Gui.lay 80
 in LoadTxt 295, 316
 Guibld.c 80
 description 83
 in Equate 83
 guibld.c
 sidebar functions 228
 Guibc.c 80
 description 83
 in AnimIPanes 318
 in Equate 83, 84
 guibc.c
 in SimpAvg 296, 297, 317
 Guibc.tmp

- description 83
- guicb.tmp
 - in SimpAvg 296, 317
- GUIDefs.c 80
 - description 82
 - in Equate 82
- GUIDEFS.h 80
 - description 82
 - in Equate 82
- GUI_TextFieldSetString function 83

H

- Help 209
 - adding to Equate Add-on 211
 - equate.html 211

I

- Images 164
- Immediate Loading 252–255
 - data journaling 254
 - no data journaling 253
- Include files
 - FORTRAN 325
- IndCurveValues array 285
- Initialization
 - of add-ons 45
- InitializeCurveParams function 294
- InitializeGUICurveParams 298
- InitTecAddOn function 26, 37, 240, 246, 307, 318
- InitTecAddOn113 21, 34, 35, 235
- InsertProbeValueInWorkingArray function 289
- Interface
 - sidebar
 - size 228
- Interpolate Data 101–106
 - inverse-distance interpolation 102
 - kriging 104
 - linear interpolation 101

J

- Journal
 - data loading 263
 - deactivate 206
 - example code 206
 - immediate data load 254
 - instructions 263
 - override loading 270
 - prerequisites 206

K

- Keywords
 - for GUI controls 229
- Kriging Interpolation 104

L

- Labels 165
- Libraries
 - libtec 24
 - linked libraries 19
 - shared libraries 19
- libtec 24
- Lighting effect 123
- Line Legends 121
- Line Plot 115–122

- axes 119
- curve types 120
- legends 121
- mapping layers 117
- Linemap 111–115
- Link, Frames 166
- Loading add-ons
 - \$!LoadAddOn command 40
 - add-ons loaded by all users 39
 - specifying a secondary add-on load file 39
 - specifying add-ons on the command line 39
 - specifying which add-ons to load 39
 - tecplot.add 39
- Load-on-demand 255–263
 - auto-load variable-on-demand 257
 - callbacks 263
 - load value-on-demand 262
 - load variable-on-demand 260
 - threads 263
- Localization 219

M

- Macro
 - recording with an add-on 201
- Macro language
 - accessing field data 75
 - augmenting with add-ons 199
- Main.c 80
 - in AnimIPlanes 323
 - in Equate 84
 - in SumProbe 240
- main.c 26, 37
 - in PolyInt 280
 - in SimpAvg 295
- Mapping Layers 117
- MenuCallback function 26, 37, 239
- Menus
 - coding for 234
- Mesh Layer 123
- MessageString
 - in Converter 308
- Metadata, *see Auxiliary Data*
- MFC DLL example 31
- Microsoft DLLs 29
- Modal dialogs
 - in Windows 221
- Modeless dialogs
 - in Windows 222
 - PreTranslateMessage function 223
- Mopup
 - of add-ons 46
- MulNum
 - about 83
- Multi-threading 59–60
 - condition variables 60
 - mutex 60
 - thread pool 59
- Multithreading
 - thread poolb 59
- Mutex 60
- MyProbeCallback function 240

N

- Node List 265

INDEX

see Connectivity List

O

- Objects
 - shared objects 19
- Objects, Derived 135
- Objects, Picked
 - operating on 214
 - pick list 215
 - types 213
- Online Help 209
- Online help
 - adding to Equate Add-on 211
 - equate.html 211
- openlink dataformat.pdf 67
- Option menus
 - special coding 234
- Options for TGB controls 229
- Ordered Data 63
 - face-neighbors 64
- Output, console 328

P

- PARAMETER Values
 - for FORTRAN glue functions 327
- Persistence 263
 - deactivate 206
 - example code 206
 - prerequisites 206
- Picked Objects
 - operating on 214
 - pick list 215
 - types 213
- Plot Style 111–174
 - animation 169
 - axes 151–153
 - blanking 165
 - coloring 146–150
 - basic colors 147
 - multi-color 147
 - rgb coloring 148
 - field plot 122–143
 - derived objects 135
 - fieldmap layers 122–143
 - fieldmap 111
 - example 114
 - styles 115
 - frame linking 166
 - geometry 162
 - images 164
 - labels 165
 - line plot 115–122
 - axes 119
 - curve types 120
 - line legends 121
 - mapping layers 117
 - linemap 111
 - example 112
 - styles 115
 - plotting data subset 143
 - points 143
 - surfaces 144
 - text 153–155
 - transient data 170
- view options 172
 - rotation 172
 - translation 173
 - zoom 173

Plotting

- animation 169
- axes 151–153
- blanking 165
- code generator 175–179
 - incorporating output 177
- launching 177
- undisplayed commands 178

coloring 146–150

- basic colors 147
- multi-color 147
- rgb coloring 148

data subset 143

- points 143
- surfaces 144

field plot 122–143

- derived objects 135
- fieldmap layers 122–143

fieldmap 111

- example 114
- styles 115

frame linking 166

labels 165

line plot 115–122

- axes 119, 120
- line legends 121
- mapping layers 117

linemap 111

- example 112
- styles 115

style 111–174

style hierarchy 175

text 153–155, 162, 164

transient data 170

view options 172

- rotation 172
- translation 173
- zoom 173

Polyhedral data

- boundary connection 70
- boundary face 70
- face neighbors 69
- facemap data 68

PolyInt add-on

- arrays 284
- DepCurveValues array 285
- IndCurveValues array 285

PolyInt function 281, 284, 287

Porting

- add-ons between Windows and UNIX 181

Porting add-ons between UNIX and Windows 181

PrepareWorkingArray function 285

PreTranslateMessage function

- for modeless dialogs 223

Probe

- value improvement 276

Q

- Quick Macro Panel 199
 - assigning macros or add-ons to buttons 199

R

- release flag 25
- Replace Data 270
- Right-hand rule
 - face neighbors 69
- Rotate Data 172
- Runmake 25
- Running your add-on 238

S

- Scatter Layer 131
- Sets
 - example 193
 - 191
- Shared Data 78
- Shared libraries 19
- Shared library
 - loading add-ons 40
- Shared objects 19
- Sidebar
 - sizing 228
- SidebarSizing** 228
- SimpAvg add-on
 - configuration 291
 - Curve-Fit dialog 301
 - dialog initialization 296
 - dialog launch 296
 - dialog work 299
 - register in Tecplot 295
- SimpAvgadd-on
 - exercises 304
- SimpleAverage function 280, 302, 303
- Skip parameter
 - in AnimIPanes 319
- Skip text field 318
- Smooth Data 94
 - limitations 94
- Source code
 - building using TGB 237
 - examples 20, 80, 239, 279, 306, 315
 - GUI source code 82
- Specifying a secondary add-on load file 39
- State change callbacks
 - in add-ons 323
- State changes
 - description 322
 - in add-ons 322
 - monitoring 322
- State variables 83
- StateChangeCallback function 323
- __stdcall
 - calling convention 328
- String lists
 - example 188
- StringList_pa
 - about 308
- Strings
 - UTF8 219
- SumProbe add-on
 - description 239
 - exercises 241
- Surfaces to plot 123
- Syntax, Equation
 - auxiliary data 93

- derivative functions 91
- difference functions 91
- share variable 94

T

- TECADDONDEVDIR 24
- TECADDONDEVPLATFORM 24
- TECADDONFILE 24, 39
- tecdev.add 40
- TecGUISidebarActivate 228
- TECGUITECPLOTSIDEBAR 228
- TecIO function 245
- Tecplot
 - register add-ons 295
 - running with add-ons 39
- Tecplot GUI Builder
 - source code 82
 - using 29
- Tecplot GUI Builder (TGB) 25
- Tecplot.add 29
- tecplot.add 39
- tecplot.cfg 39
- tecplot.fnt 39
- TecUtil function 239
- TecUtil functions 308
- TecUtilCurveRegisterExtCrvFit function 280
- TecUtilCurveSetExtendedSettings function 294
- TecUtilDialogGetVariables function 240
- TecUtilImportAddLoader function 246
- TecUtilMenuAddOption function 26, 37, 240
- TecUtilProblInstallCallback function 240
- TecUtilStringAlloc function 292
- TecUtilTecDat function 308
- TecUtilTecEnd function 308
- TecUtilTecIni function 308
- TecUtilTecZne function 308
- TecUtilVarIsEnabled function 85
- TecUtilZonelsEnabled function 85
- Text 153–155
- TGB
 - building source code building 237
 - used with Developer Studio 225
- TGB control options 229
- TGB created files 237
- Thread mutex 60
- Thread pool 59
- Threading 59–60
 - condition variables 60
 - mutex 60
 - thread pool 59
- _token variable 308
- Transform Coordinates 95
- Transient Data 170
 - display 170
 - loading 269
- Translation, view options 173
- Triangulation, irregular data point 105
- Troubleshooting add-ons 31, 35
- Types of controls and keywords 229

U

- UNIX
 - porting add-ons to Windows 181
- UpdateMainDialogSensitivities function 298

INDEX

Using sets 191
UTF8 219

V

Variables
 CurveSettings 284
 XYMapNum 284
Vector Layer 129
Visualization
 animation 169
 axes 151–153
 blanking 165
 code generator 175–179
 incorporating output 177
 launching 177
 undisplayed commands 178
 coloring 146–150
 basic colors 147
 multi-color 147
 rgb coloring 148
 field plot 122–143
 derived objects 135
 fieldmap layers 122–143
 fieldmap 111
 example 114
 styles 115
 frame linking 166
 labels 165
 line plot 115–122
 axes 119
 curve types 120
 line legends 121
 mapping layers 117
 linemap 111
 example 112
 styles 115
 plot style 111–174
 plotting data subset 143
 points 143
 surfaces 144
 style hierarchy 175
 text 153–155, 162, 164
 transient data 170
 view options 172
 rotation 172
 translation 173
 zoom 173

W

Windows
 how to build add-ons 29
 porting add-ons to UNIX 181
Write statements
 to console 328

X

XYDataPointsCallback function 284, 285, 287, 301
XYMapNum parameter 293
XYMapNum variable 284
XY-Plot Curve Info dialog 290

Z

Zone
 see Fieldmap

Zone effects
 lighting 123
Zone Style dialog 295, 300
ZoneList text field 318
 initialization 318
ZoneSet
 in AnimIPanes 320
ZoneSet parameter
 in AnimIPanes 319
Zoom, view option 173