SC34-3262-02

IBM Visualization Data Explorer

QuickStart Guide

Version 3 Release 1 Modification 4

IBM Visualization Data Explorer

SC34-3262-02

QuickStart Guide

Version 3 Release 1 Modification 4

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xi.

Third Edition (September 1997)

This edition applies to IBM Visualization Data Explorer Version 3.1.4, to IBM Visualization Data Explorer SMP Version 3.1.4, and to all subsequent releases and modifications thereof until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product. Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
Thomas J. Watson Research Center/Hawthorne
Data Explorer Development
P.O. Box 704
Yorktown Heights, NY 10598-0704
USA

If you send information to IBM, you grant IBM a nonexclusive right to use or distribute that information, in any way it believes appropriate, without incurring any obligation to you.

© Copyright International Business Machines Corporation 1991-1997. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Figures . . . vii

Tables . . . ix

Notices . . . xi
Products, Programs, and Services . . . xii
Trademarks and Service Marks . . . xii
Copyright notices . . . xiii

About This Guide . . . xix
A Suggestion on Where to Start . . . xx
A Note on the Window System . . . xx
Typographic Conventions . . . xxi
Related Publications and Sources . . . xxi
  IBM Publications . . . xxi
  Non-IBM Publications . . . xxi
  Other sources of information . . . xxii

Chapter 1. A Very Quick Overview . . . 1

Chapter 2. Tutorial I: Using Data Explorer . . . 3
2.1 Starting Data Explorer . . . 4
2.2 Accessing the Tutorials . . . 5
2.3 Where To Begin . . . 6
2.4 Opening and Executing a Visual Program . . . 6
  Opening a Visual Program . . . 6
  Executing a Visual Program . . . 8
2.5 Controlling the Appearance of an Object: The Image Window . . . 10
  Size, View, and AutoAxes . . . 10
  Using the Sequencer . . . 14
  Using Control Panels . . . 15
  Using the Colormap Editor . . . 16

Chapter 3. Tutorial II: Editing and Creating Visual Programs . . . 21
3.1 Editing a Visual Program: The Basics . . . 22
3.2 Creating a Visual Program: Two short examples . . . 24
  A simple two-dimensional field . . . 24
  A simple three-dimensional field . . . 24
3.3 Importing Data . . . 25
  Example 1 . . . 25
  Example 2 . . . 26
3.4 A thumbnail Sketch of the Data Prompter Choices . . . 27
3.5 Importing Your Own Data . . . 28
3.6 Visualizing 2-Dimensional Data . . . 29
  Colors . . . 29
  Contour Lines . . . 29
  Streamlines . . . 30
  RubberSheet . . . 30
  2-D Scalar Glyphs . . . 31
  2-D Vector Glyphs . . . 32

© Copyright IBM Corp. 1991-1997

3.7 Visualizing 3-Dimensional Data 32
Isosurfaces 32
Slices 32
Streamlines 33
3-D Scalar Glyphs 34
3-D Vector Glyphs 34
Volume Rendering 34
3.8 Tasks and Tools 36
Adding Captions 36
Adding Input Tabs to Tool Icons 37
Connecting Scattered Data Points 37
Controlling Execution with Switch 37
Controlling Inputs: Configuration Dialog Boxes 38
Controlling Inputs: Interactors 39
Creating Animations 41
Creating and Using Macros 42
Data-driven Tools 43
Modules: Using AutoColor 44
Modules: Using Compute 44
Modules: Using Map 45
Modules: Using Plot 46
Processing Images 46
Saving and Printing Images 47
3.9 Scripting Language 47

Chapter 4. Sample Visual Programs and Sample Macros 49
4.1 Sample Visual Programs 49
Simple Visual Programs 49
2-Dimensional Data 49
3-Dimensional Data 50
Annotation 52
Categorical 53
Colormap Editor 53
Compute 54
Data-driven Interactors 54
Debugging 54
Distributed Processing 54
Image Processing 54
Importing Data 55
Interface Control 55
Looping 55
Miscellaneous 56
Probes 57
Rendering 57
Scattered Data 58
Sequencer 58
4.2 Sample Macros 59

Chapter 5. Importing Data 61
5.1 General Array Importer 63
Describing the Data 63
Creating a Header File 66
Some Notes on General Array Importer Format 66
5.2 Importing Data: Header File Examples 67

Record Style: Single-Variable Data 67
Record Style: Multivariable Data 75
Columnar Style 81
5.3 Header File Syntax: Keyword Statements 85
file 86
grid 87
points 87
block 88
dependency 88
field 88
format 89
header 89
interleaving 90
layout 92
majority 92
recordseparator 92
series 93
structure 94
type 94
positions 94
end 96
5.4 Data Prompter 96
For Future Reference 97
Supported Formats 97
Initial Dialog Box 98
Simplified Data Prompter 101
Full Data Prompter 103
5.5 Data Prompter Browser 109
Starting the Browser 109
Browser Menu Bar 109
Browser Text Window 111
Browser Offset Area 111
5.6 Using the Header File to Import Data 111

Glossary 113

Index 117

Figures

1. Main features of Data Explorer . . . 1
2. Startup Data Explorer Window . . . 4
3. File selection dialog box . . . 5
4. Visual Program Editor Window (VPE) . . . 7
5. Image Window . . . 9
6. View Control Dialog Box . . . 11
7. Sequence Control Panel . . . 14
8. Control Panel with two Interactors. . . . 15
9. Colormap Editor for .../example1.net . . . 18
10. Add Control Points Dialog Box . . . 19
11. Examples of Grid Types . . . 64
12. Examples of Data Dependency . . . 64
13. Row- versus Column-Majority Grids . . . 65
14. Block and Columnar Styles of Data Organization . . . 66
15. Initial Data Prompter window . . . 99
16. Simplified Data Prompter . . . 102
17. Full Data Prompter . . . 105
18. File Browser Window . . . 110
19. Import Configuration Dialog Box . . . 112

© Copyright IBM Corp. 1991-1997

Tables

1. Image Characteristics Controlled by the Colormap Editor 17

© Copyright IBM Corp. 1991-1997

Notices

Products, Programs, and Services xii
Trademarks and Service Marks xii
Copyright notices xiii

© Copyright IBM Corp. 1991-1997

Products, Programs, and Services

References in this publication to IBM* products, programs, or services do not imply that IBM intends to make these available in all countries in which it operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give the user any license to those patents. License inquiries should be sent, in writing, to:

International Business Machines Corporation
IBM Director of Licensing
500 Columbus Avenue
Thornwood, New York 10594
USA

Trademarks and Service Marks

The following terms, marked by an asterisk (*) at their first occurrence in this publication, are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries.

AIX
IBM
IBM Power Visualization System
RISC System/6000
Visualization Data Explorer

The following terms, marked by a double asterisk (**) at their first occurrence in this publication, are trademarks of other companies.

AViiON	Data General Corporation
DEC	Digital Equipment Corporation
DGC	Data General Corporation
Graphics Interchange Format (GIF)	CompuServe, Inc.
Hewlett-Packard	Hewlett-Packard Company
HP	Hewlett-Packard Company
iFOR/LS	Apollo Computer, Inc.
Motif	Open Software Foundation
NetLS	Apollo Computer, Inc.
Network Licensing Software	Apollo Computer, Inc.
OpenWindows	Sun Microsystems, Inc.
OSF	Open Software Foundation, Inc.
PostScript	Adobe Systems, Inc.
X Window System	Massachusetts Institute of Technology

Copyright notices

IBM Visualization Data Explorer contains software copyrighted as follows:

- E. I. du Pont de Nemours and Company

  © Copyright 1997 E. I. du Pont de Nemours and Company

  Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of E. I. du Pont de Nemours and Company not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. E. I. du Pont de Nemours and Company makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

  E. I. du Pont de Nemours and Company disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness, in no event shall E. I. du Pont de Nemours and Company be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

- National Space Science Data Center

  © Copyright 1990-1994 NASA/GSFC

  National Space Science Data Center
  NASA/Goddard Space Flight Center
  Greenbelt, Maryland 20771 USA
  (NSI/DECnet -- NSSDCA::CDFSUPPORT)
  (Internet -- CDFSUPPORT@NSSDCA.GSFC.NASA.GOV)

- University Corporation for Atmospheric Research/Unidata

  © Copyright 1993, University Corporation for Atmospheric Research

  Permission to use, copy, modify, and distribute this software and its documentation for any purpose without fee is hereby granted, provided that the above copyright notice appear in all copies, that both that copyright notice and this permission notice appear in supporting documentation, and that the name of UCAR/Unidata not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. UCAR makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty. It is provided with no support and without obligation on the part of UCAR Unidata, to assist in its use, correction, modification, or enhancement.

- NCSA

  NCSA HDF version 3.2r4
  March 1, 1993

  NCSA HDF Version 3.2 source code and documentation are in the public domain. Specifically, we give to the public domain all rights for future licensing of the source code, all resale rights, and all publishing rights.

We ask, but do not require, that the following message be included in all derived works:

Portions developed at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, in collaboration with the Information Technology Institute of Singapore.

THE UNIVERSITY OF ILLINOIS GIVES NO WARRANTY, EXPRESSED OR IMPLIED, FOR THE SOFTWARE AND/OR DOCUMENTATION PROVIDED, INCLUDING, WITHOUT LIMITATION, WARRANTY OF MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE

- Gradient Technologies, Inc. and Hewlett-Packard Co.

  © Copyright Gradient Technologies, Inc. 1991,1992,1993
  © Copyright Hewlett-Packard Co. 1988,1990

  June, 1993

  UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

  Gradient is a registered trademark of Gradient Technologies, Inc.

  NetLS and Network Licensing System are trademarks of Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.

- Sam Leffler and Silicon Graphics

  © Copyright 1988-1996 Sam Leffler
  © Copyright 1991-1996 Silicon Graphics, Inc.

  Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

  THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

  IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- Compuserve Incorporated

  The Graphics Interchange Format © is the copyright property of Compuserve Incorporated. GIF(SM) is a Service Mark property of Compuserve Incorporated.

- Integrated Computer Solutions, Inc.

  Motif Shrinkwrap License

  READ THIS LICENSE AGREEMENT CAREFULLY BEFORE USING THE PROGRAM TAPE, THE SOFTWARE (THE "PROGRAM"), OR THE ACCOMPANYING USER DOCUMENTATION (THE "DOCUMENTATION").

THIS AGREEMENT REPRESENTS THE ENTIRE AGREEMENT CONCERNING THE PROGRAM AND DOCUMENTATION POSAL, REPRESENTATION, OR UNDERSTANDING BETWEEN THE PARTIES WITH RESPECT TO ITS SUBJECT MATTER. BY BREAKING THE SEAL ON THE TAPE, YOU ARE ACCEPTING AND AGREEING TO THE TERMS OF THIS AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND NY THE TERMS OF THIS AGREEMENT, YOU SHOULD PROMPTLY RETURN THE CONTENTS, WITH THE TAPE SEAL UNBROKEN; YOUR MONEY WILL BE REFUNDED.

1. License: ISC remains the exclusive owner of the Program and the Documentation. ICS grant to Customer a nonexclusive, nontransferable (except as provided herein) license to use, modify, have modified, and prepare and have prepared derivative works of the Program as necessary to use it.

2. Customer Rights: Customer may use, modify and have modified and prepare and have prepared derivative works of the Program in object code form as is necessary to use the Program. Customer may make copies of the Program up to the number authorized by ICS in writing, in advance. There shall be no fee for Statically linked copies of the Motif libraries. Statically linked copies are object code copies integrated within a single application program and executable only with that single application. Run Time copies require payment of ICS' then applicable fee. Run Time copies are copies which include any portion of a linkable object file (“.o” file), library file (“.a” file), the window manager (mwm manager), the U.I.L. compiler, a shared library, or any tool or mechanism that enables generation of any portion of such components; other copies will require payment of ICS' applicable fees. TRANSFERS TO THIRD PARTIES OF COPIES OF THE LICENSED PROGRAMS, OR OF APPLICATIONS PROGRAMS INCORPORATING THE PROGRAM (OR ANY PORTION THEREOF), REQUIRE ICS' RESELLER AGREEMENT. Customer may not lease or lend the Program to any party. Customer shall not attempt to reverse engineer, disassemble or decompile the program.

3. Limited Warranty: (a) ICS warrants that for thirty (30) days from the delivery to Customer, each copy of the Program, when installed and used in accordance with the Documentation, will conform in all material respects to the description of the Program's operations in the Documentation. (b) Customer's exclusive remedy and ICS' sole liability under this warranty shall be for ICS to attempt, through reasonable efforts, to correct any material failure of the Program to perform as warranted, if such failure is reported to ICS within the warranty period and Customer, at ICS' request, provides ICS with sufficient information (which may include access to Customer's computer system for use of Customer's copies of the Program by ICS personnel) to reproduce the defect in question; provided, that if ICS is unable to correct any such failure within a reasonable time, ICS may, at its sole option, refund to the Customer the license fee paid for the Product. (c) ICS need not treat minor discrepancies in the Documentation as errors in the Program, and may instead furnish correction to the Program. (d) ICS does not warrant that the operation of the Program will be uninterrupted or error-free, or that all errors will be corrected. (e) THE FOREGOING WARRANTY IS IN LIEU OF, AND ICS DISCLAIMS, ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL ICS BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT

LIMITATION LOST PROFITS, ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM OR DOCUMENTATION.

4. Term and Termination: The term of this agreement shall be indefinite; however, this Agreement may be terminated by ICS in the event of a material default by Customer which is not cured within thirty (30) days after the receipt of notice of such breech by ICS. Customer may terminate this Agreement at any time by destruction of the Program, the Documentation, and all other copies of either of them. Upon termination, Customer shall immediately cease use of, and return immediately to ICS, all existing copies of the Program and Documentation, and cease all use thereof. All provisions hereof regarding liability and limits thereon shall survive the termination of this the Agreement.

5. U.S. GOVERNMENT LICENSES. If the Product is provided to the U.S. Government, the Government acknowledges receipt of notice that the Product and Documentation were developed at private expense and that no part of either of them is in the public domain. The Government acknowledges ICS' representation that the Product is “Restricted Computer Software” as defined in clause 52.227-19 of the Federal Acquisition Regulations (the “FAR” and is “Commercial Computer Software” as defined in Subpart 227.471 of the Department of Defense Federal Acquisition Regulation Supplement (the “DFARS”). The Government agrees that (i) if the software is supplied to the Department of Defense, the software is classified as “Commercial Computer Software” . and that the Government is acquiring only “Restricted Rights” in the software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS and (ii) if the software is supplied to any unit or agency of the Government other than the Department of Defense, then notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Clause 52.227-19(c)(2) of the FAR. All copies of the software and the documentation sold to or for use by the Government shall contain any and all notices and legends necessary or appropriate to assure that the Government acquires only limited right in any such documentation and restricted rights in any such software.

6. Governing Law: This license shall be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts as a contract made and performed therein.

- OMRON Corporation, NTT Software Corporation, and MIT

© Copyright 1990, 1991 by OMRON Corporation, NTT Software Corporation,
and Nippon Telegraph and Telephone Corporation
© Copyright 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of OMRON, NTT Software, NTT, and M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. OMRON, NTT Software, NTT, and M.I.T. make no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

OMRON, NTT SOFTWARE, NTT, AND M.I.T. DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL OMRON, NTT SOFTWARE, NTT, OR M.I.T. BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

About This Guide

A Suggestion on Where to Start xx
A Note on the Window System xx
Typographic Conventions xxi
Related Publications and Sources xxi
  IBM Publications xxi
  Non-IBM Publications xxi
  Other sources of information xxii

© Copyright IBM Corp. 1991-1997

This guide presents a "hands on" introduction to Data Explorer and is designed to help you start working with it immediately.

- Chapter 1, "A Very Quick Overview" on page 1, is a 2-page summary of the Data Explorer system and its main features. The purpose of this overview is to give you a clear general outline of the Data Explorer tools and interfaces and of their functions in the process of visualizing data.
- Chapter 2, "Tutorial I: Using Data Explorer" on page 3, covers most of the basic functions of Data Explorer—at a level accessible to any new user. It briefly introduces the Visual Program Editor (VPE) before concentrating on the Image Window—the user interface for directly presenting and manipulating the images generated by visual programs.
- Chapter 3, "Tutorial II: Editing and Creating Visual Programs" on page 21, uses step-by-step examples to illustrate the use of the VPE in the more advanced tasks of creating and modifying visual programs. (Chapter 3, "Tutorial II: Editing and Creating Visual Programs" on page 21 also includes a list of the sample visual programs supplied with Data Explorer.)
- Chapter 5, "Importing Data" on page 61, explains how to import data into Data Explorer, using the Data Prompter and its companion file-viewing facility, the Data Browser.

A Suggestion on Where to Start

If you have never used Data Explorer before, read Chapter 1, "A Very Quick Overview" on page 1 and then start on the tutorials. If you are familiar with Data Explorer, you may want to focus on sample programs that involve the kind of visualization tasks you encounter in your own work. For that purpose, Chapter 4, "Sample Visual Programs and Sample Macros" on page 49, lists the sample visual programs and macros supplied with Data Explorer.

You can follow the tutorials in this Guide or on-line. To use the on-line version,

- type `dx -tutor` and press the Enter key, *or*
- Choose **Run Tutorial** from the Data Explorer Startup window.

A Note on the Window System

This Guide assumes that you have some familiarity with your operating system, with the X Window System** being used, and with OSF**/Motif**. For more information, if needed, consult the appropriate window system documentation.

Any reference to the X Window System means any window server that supports the X11 protocol, including Sun's OpenWindows**.

The Motif window manager (mwm) has been used in some figures and examples in this Guide. If you are using another window manager, the title bars and window borders may differ slightly from those shown.

Typographic Conventions

Boldface	Identifies commands, keywords, files, directories, messages from the system, and other items whose names are defined by the system.
Italic	Identifies parameters with names or values to be supplied by the user.
`Monospace`	Identifies examples of specific data values and text similar to what you might see displayed or might type at a keyboard or that you might write in a program.

Related Publications and Sources

IBM Publications

- *IBM Visualization Data Explorer User's Guide*, SC38-0496

  Details the main features of Data Explorer, including the data model, data import, the user interface, the Image window, and the visual program editor. and the scripting language. Of particular interest to programmers: chapters on the data model and the scripting language.

- *IBM Visualization Data Explorer User's Reference*, SC38-0486

  Contains detailed descriptions of Data Explorer's tools.

  Note: Consult this reference if you are creating visual programs or scripts.

- *IBM Visualization Data Explorer Programmer's Reference*, SC38-0497

  Contains detailed descriptions of the Data Explorer library routines.

  Note: Consult this reference if you are writing your own modules for Data Explorer.

Non-IBM Publications

The following treat various aspects of computer graphics and visualization:

Adobe Systems Incorporated, *PostScript Language Reference Manual*, 2nd Ed., Addison-Wesley Publishing Company, Massachusetts, 1990.

Aldus Corporation and Microsoft Corporation, *Tag Image File Format Specification, Revision 5.0*, Aldus Corporation, Washington, 1988.

Arvo, Jim, ed., *Graphics Gems II*, Academic Press, Inc., Boston, Massachusetts, 1991.

Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company; Massachusetts, 1990.

Friedhoff, Richard M., and Benzon, William, *Visualization: The Second Computer Revolution*, New York, Harry N. Abrams, Inc., 1989.

Glassner, Andrew, ed., *Graphics Gems*, Academic Press, Inc., Boston, Massachusetts, 1990.

Hill, F.S., Jr., *Computer Graphics*. Macmillan Publishing Company, New York, 1990.

Kirk, David, ed., *Graphics Gems III*, Academic Press, Inc., Boston, Massachusetts, 1992.

Robin, Harry, *The Scientific Image: from cave to computer*, Harry N. Abrams, Inc., New York, 1992.

Rogers, David F., *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, New York, 1985.

Rogers, David F. and Adams, J.Alan, *Mathematical Elements for Computer Graphics*, 2nd Ed., New York, McGraw-Hill Book Company, 1990.

SIGGRAPH Conference Proceedings, Association for Computing Machinery, Inc.: A Publication of ACM SIGGRAPH, New York, various years.

Tufte, Edward, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut, 1983.

Other sources of information

For additional ideas, consult the “DX Repository,” available through anonymous FTP (`ftp.tc.cornell.edu.` in directory `pub/Data.Explorer`), and gopher (`ftp.tc.cornell.edu.` port 70). This public software resource includes information and visual programs contributed by Data Explorer users from around the world. We encourage you to contribute your innovations and ideas to the Repository, in the form of new modules, macros, visual programs, and tips and tricks you discover as you learn and master Data Explorer.

On the Internet, the newsgroup `comp.graphics.apps.data-explorer` is used by customers around the word to share information and ask questions. This newsgroup is also followed by Data Explorer developers.

If you have access to the World Wide Web, you can find the Data Explorer home page at `http://www.almaden.ibm.com/dx/`.

Chapter 1. A Very Quick Overview

Data Explorer is a system of tools and user interfaces for visualizing data. In general terms the visualization of data can be considered a 3-stage process:

1. Describing and importing data
2. Processing the data through a visualization program
3. Presenting the resulting image.

The simple diagram below shows where the chief tools and interfaces of Data Explorer fit into this process. The numbered list that follows Figure 1 is keyed to the diagram. The list items briefly describe the roles of these features in Data Explorer visualization and give references to the relevant Data Explorer documentation.

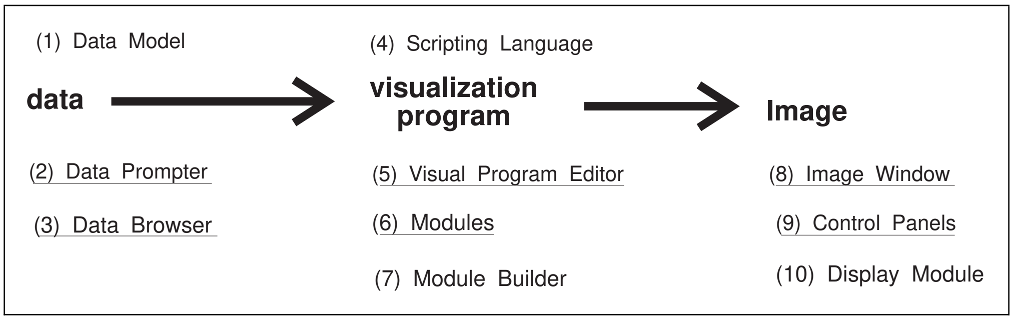


Figure 1. Main features of Data Explorer. Tools and interfaces are positioned to show their general place in the visualization process. All those listed below the data→image sequence are interactive interfaces except for the “modules” (6 and 10), which are directly accessible and manipulable in the Visual Program Editor (5). Numbers correspond to those in the accompanying list. Underlined features are discussed in this Guide. Features not underlined are discussed in other documentation, cited in the list.

(1) Data Model

The set of definitions, rules, and conventions used to describe Data Explorer entities (including data fields, geometrical objects, and images).

See *IBM Visualization Data Explorer User’s Guide*.

(2) Data Prompter

A user interface for describing data to be imported into Data Explorer.

See 3.3, “Importing Data” on page 25 and 5.4, “Data Prompter” on page 96 in this Guide.

(3) Data Browser

A user interface for viewing a data file, determining the layout and organization of the data it contains, and transferring this information to the Data Prompter (2).

See 5.5, “Data Prompter Browser” on page 109 in this Guide.

(4) Scripting Language

A simple, high-level language for creating visualization programs. It can also be used directly in a command mode to perform various tasks. Visual programs—i.e., the visualization programs displayed in the Visual Program Editor window (5) as “networks” of module icons—are also written in the scripting language. A visual program constructed in this window by the user is translated into the same language when it is saved to disk.

© Copyright IBM Corp. 1991-1997

See *IBM Visualization Data Explorer User's Guide*.

(5) Visual Program Editor (VPE)

A graphical user interface for creating and modifying visual programs (networks). Programs created with this editor are translated into the scripting language (4) by Data Explorer and are stored in that form.

See 3.1, "Editing a Visual Program: The Basics" on page 22 in this Guide; see also *IBM Visualization Data Explorer User's Guide*.

(6) Modules

The "building blocks" (visualization "tools") that constitute a visual program network. They can be directly accessed and manipulated in the Visual Program Editor (5).

See *IBM Visualization Data Explorer User's Reference*.

(7) Module Builder

A user interface for creating customized modules to be used in visual programs.

See *IBM Visualization Data Explorer Programmer's Reference*.

(8) Image Window

An interactive window for viewing and modifying the presentation of the image produced by a visual program.

See 2.5, "Controlling the Appearance of an Object: The Image Window" on page 10 in this Guide.

(9) Control Panels

A user interface for changing the parameter values used by a visual program.

See "Using Control Panels" on page 15 in this Guide; see also *IBM Visualization Data Explorer User's Guide*.

(10) Display Module

An alternative to the Image window (8).

See *IBM Visualization Data Explorer User's Reference*.

Chapter 2. Tutorial I: Using Data Explorer

2.1 Starting Data Explorer 4
2.2 Accessing the Tutorials 5
2.3 Where To Begin 6
2.4 Opening and Executing a Visual Program 6
  Opening a Visual Program 6
  Executing a Visual Program 8
2.5 Controlling the Appearance of an Object: The Image Window 10
  Size, View, and AutoAxes 10
  Using the Sequencer 14
  Using Control Panels 15
  Using the Colormap Editor 16

© Copyright IBM Corp. 1991-1997

2.1 Starting Data Explorer

To start Data Explorer on a workstation:

1. Log on to your workstation.
2. Make sure that the X Window system is running with Motif or the appropriate window manager.
3. Enter on the command line:

   ```
   dx
   ```

   The Startup Data Explorer window will appear, as shown in Figure 2.

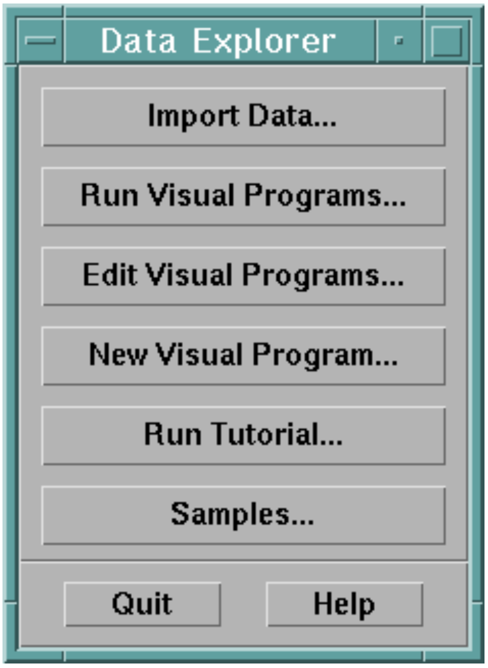
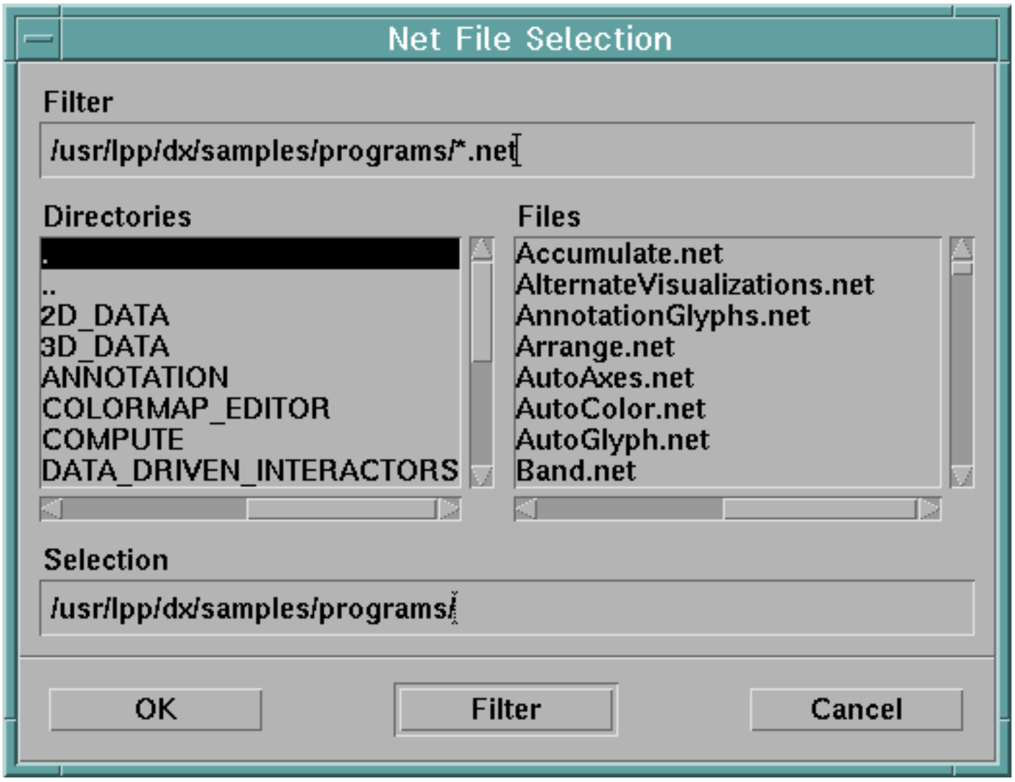

Data Explorer
Import Data...
Run Visual Programs...
Edit Visual Programs...
New Visual Program...
Run Tutorial...
Samples...
Quit
Help

Figure 2. Startup Data Explorer Window

From the Startup window you can choose to import data, run previously written visual programs, create or edit visual programs, run the Data Explorer tutorial, or run one of a set of sample programs.



Filter: /usr/lpp/dx/samples/programs/*.net

Directories:
.
..
2D_DATA
3D_DATA
ANNOTATION
COLORMAP_EDITOR
COMPUTE
DATA_DRIVEN_INTERACTORS

Files:
Accumulate.net
AlternateVisualizations.net
AnnotationGlyphs.net
Arrange.net
AutoAxes.net
AutoColor.net
AutoGlyph.net
Band.net

Selection: /usr/lpp/dx/samples/programs/

Figure 3. File selection dialog box

For Future Reference: Context-Sensitive Help

Context-Sensitive Help is available in the **Help** pull-down menu of the VPE window. Selecting this item creates a cursor in the shape of a question mark, which can then be placed on a tool icon or other feature in the window. Releasing the mouse button invokes the appropriate Help window.

2.2 Accessing the Tutorials

Tutorials I and II are both on line and can be accessed in one of three ways:

- Start Data Explorer with the command:

  ```
  dx
  ```

  then click on **Run Tutorial** in the Data Explorer Startup window (see Figure 2 on page 4).. (It is assumed for the rest of the tutorial that this is the method you have used.)
- Start Data Explorer with the command:

  ```
  dx -tutor
  ```
- Choose **Tutorial** in the Help menu of the Visual Program Editor.

In the Tutorial window, you can:

- Directly access any topic, subtopic, *sub*subtopic, or related item that is “boxed”, simply by clicking on the boxed area.
- Scroll through information that extends beyond the length of the window (using the vertical scroll bar on the right side of the window).

- Return to a topic that you have viewed during the current session, by moving the mouse cursor into the tutorial window and holding the *right-hand* mouse button: A list of the topics you have viewed appears. While holding the mouse button, move the pointer to the desired topic in the list and release the button.
- Return to the previous topic, by clicking on **Go Back** at the bottom of the tutorial window.
- Exit the tutorial by clicking on **Quit** at the bottom of the tutorial window.

2.3 Where To Begin

Tutorials I and II are designed so that you can proceed from one topic to another in whatever order you choose.

If you have not used Data Explorer before, we recommend that you start here with Tutorial I, which will quickly introduce you to many of the basic features of the user interface, as well as to many of the most commonly used Data Explorer functions.

If you already know how to run Data Explorer and want more information on how to edit and create visual programs, you can start with Chapter 3, “Tutorial II: Editing and Creating Visual Programs” on page 21 You may also start directly with any of the topics listed at the beginning of that tutorial.

If you need to learn how to start Data Explorer, see 2.1, “Starting Data Explorer” on page 4.

When you have completed Tutorial I, you should have no difficulty:

- Running a visual program and manipulating objects in the Image window.
- Exploring Data Explorer functions.

2.4 Opening and Executing a Visual Program

This section will show you how to:

- Open a visual program.
- Execute that program to display an object in the Image window.

Opening a Visual Program

A visual program must be opened before it can be executed:

1. From the Data Explorer Startup window choose **Edit Visual Program**. A file selection dialog box appears (see Figure 3 on page 5). You can also get to the file selection dialog by choosing **Run Visual Program** in the Startup window or, once the visual program editor or image window has appeared, by selecting **File** in the menu bar of the window and then selecting **Open Program**.
2. *Double click* on any part of the **Filter** field at the top of the dialog box. Any text in that field will appear against the background of a black highlight bar.
3. Type

   ```
   /usr/lpp/dx/samples/tutorial/*.net
   ```

   (The cursor automatically moves to the beginning of the text line as soon as you start typing, and the highlighted text and highlight bar disappear.)

Note: The file extension `.net` (for network) is the default for Data Explorer visual programs. In Data Explorer, “network” is a term for visual program.

4. Press **`Enter`** or click on the **`Filter`** pushbutton at the bottom of the dialog box. The **`Selection`** field just above this button now displays the directory path name. An alphabetized list of the `.net` files in the directory appears in the **`Files`** field.

   Note: If the path name you typed is invalid (the wrong list or no list of files appears), you can edit it: position the mouse cursor at the point where you want to start editing and click *once* before beginning to type.

   If the highlight bar reappears, retyping the entire path name is still unnecessary: again, position the cursor and click *once*. The highlight bar disappears, but the path name remains, with the cursor at the selected position.

5. Use the scroll bar to move to the bottom of the list in the **`Files`** field. Click on `example1.net` to highlight the file name. The **`Selection`** field now displays the file’s full path name.

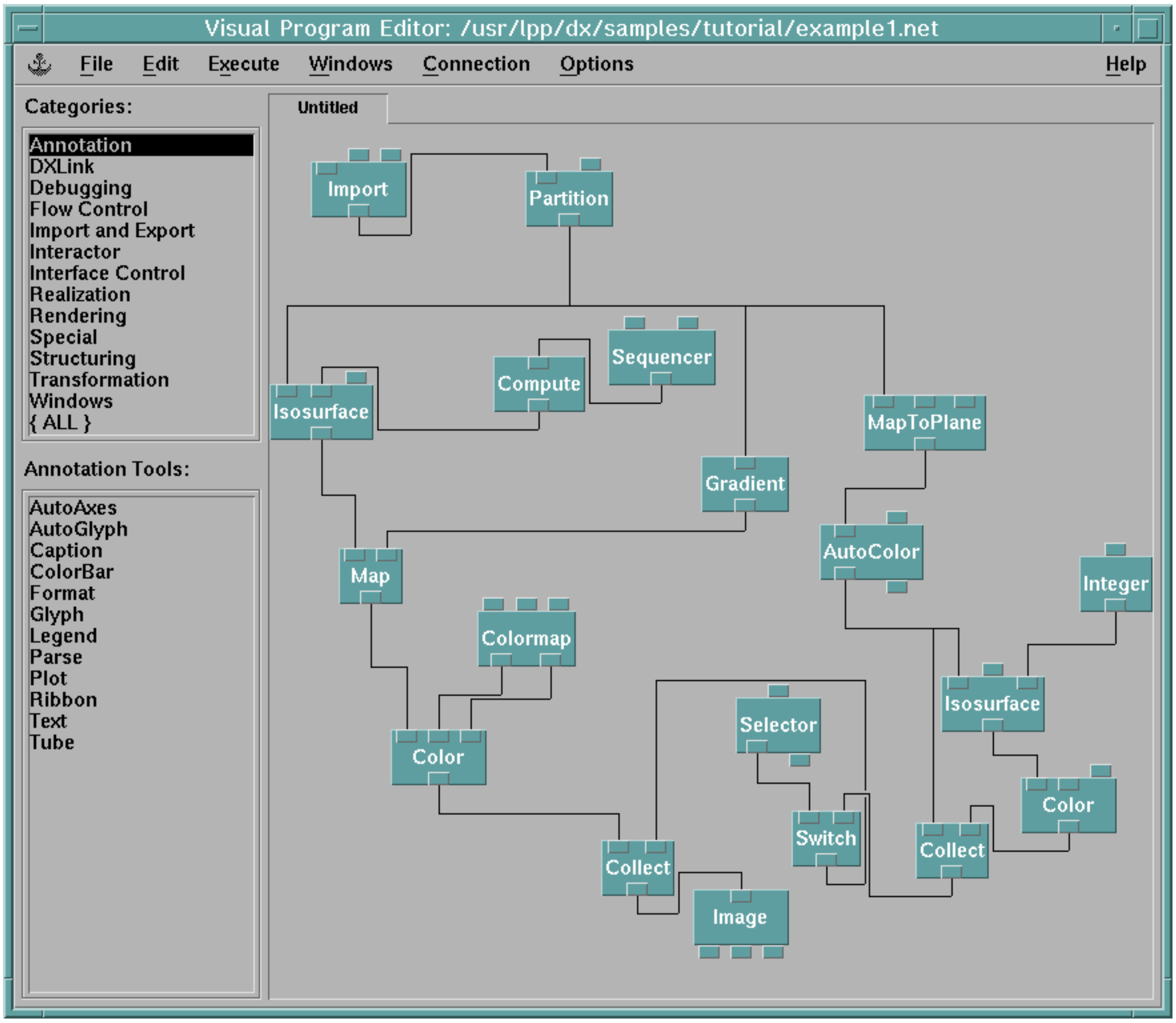

Visual Program Editor: /usr/lpp/dx/samples/tutorial/example1.net
File
Edit
Execute
Windows
Connection
Options
Help
Categories:
Annotation
DXLink
Debugging
Flow Control
Import and Export
Interactor
Interface Control
Realization
Rendering
Special
Structuring
Transformation
Windows
{ ALL }
Annotation Tools:
AutoAxes
AutoGlyph
Caption
ColorBar
Format
Glyph
Legend
Parse
Plot
Ribbon
Text
Tube
Untitled
Import
Partition
Sequencer
Compute
Isosurface
MapToPlane
Gradient
AutoColor
Integer
Map
Colormap
Selector
Isosurface
Color
Color
Switch
Collect
Collect
Image

Figure 4. Visual Program Editor Window (VPE). This window consists of two palettes and a “canvas.” The palettes list categories of tools (or modules, top left) and individual tools in a selected category (bottom left). The canvas is the area for creating and editing visual programs (“networks”). as shown here, a network consists of tool (module) icons and connecting lines (“arcs”). The name of the network program appears in the title bar at the top of the window. (Use of the VPE is described in Chapter 3, “Tutorial II: Editing and Creating Visual Programs” on page 21.)

6. Click on **OK** to open the `example1.net` visual program: the dialog box closes; a "network" of modules and connecting lines appears in the "canvas" area of the VPE window; and the full path name of the program appears in the title bar of the window. Opening is completed.

For a short description of `example1.net`, click on **Help** at the right side of the menu bar and select **Application Comment** in the pull-down menu.

For Future Reference

- Double clicking on a name in the **Files** field also opens a program.
- It is not necessary to clear the canvas of one program before opening another. Just open the new program.
- If for any reason you want to start over again with a clean canvas—without closing Data Explorer: click on **File** in the menu bar and select **New** in the pull-down menu. This sequence clears the canvas of the previous program network and any associated windows.

Executing a Visual Program

To execute a visual program like `example1.net` (once it has been opened):

1. Click on **Execute** in the menu bar of the VPE window.

   Note: A visual program can be executed from any window that has **Execute** in its menu bar.

2. Select **Execute Once** in the pull-down menu. Execution (which may take several seconds) is indicated by the highlighted **Execute** in the menu bar and the brief highlighting of various icons in the canvas area. The window that appears displays the image of a water molecule (Figure 5 on page 9).

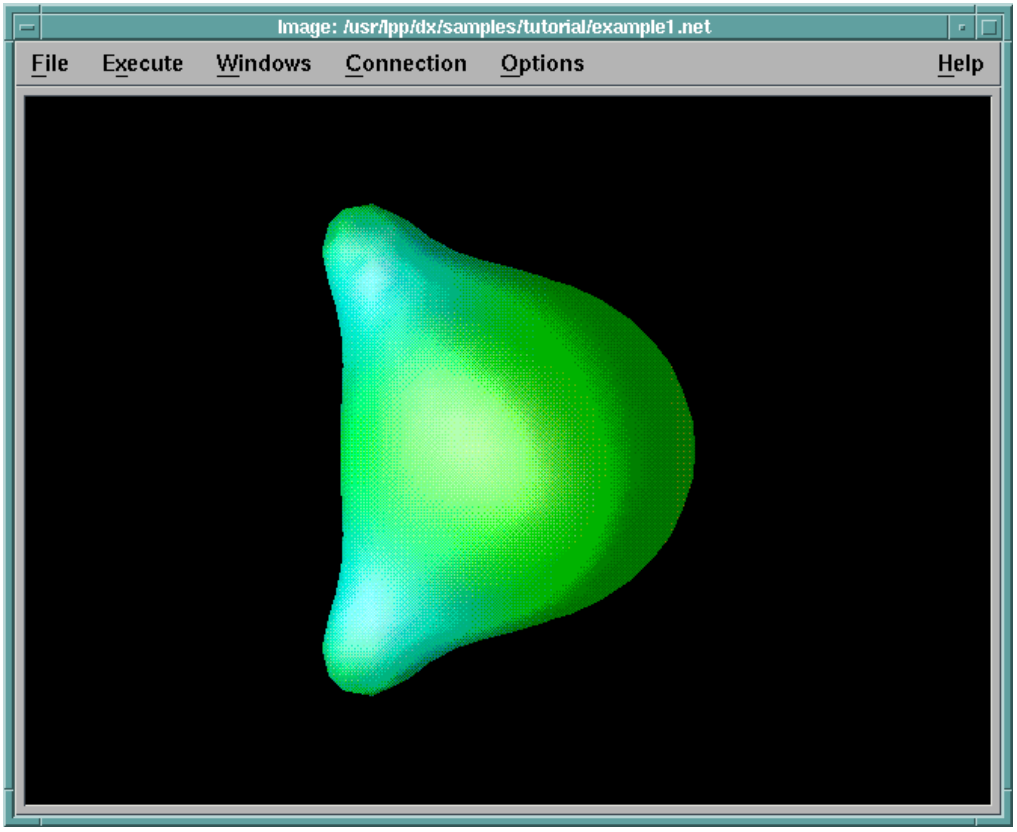


Figure 5. Image Window. The window shown here displays the image generated by the visual program /usr/lpp/dx/samples/tutorial/example1.net.

Once an object appears in the Image window, you can do a number of things with it, including:

- Changing the view of the object (see 2.5, "Controlling the Appearance of an Object: The Image Window" on page 10).
- Editing the visual program that generates the image (see 3.1, "Editing a Visual Program: The Basics" on page 22).
- Saving and Printing its image, using the **File** pull-down menu in the menu bar (see "Saving and Printing Images" on page 47).

For Future Reference

1. Many of the options in pull-down menus can be invoked by "accelerator" keyboard sequences. In the menus, these "shortcuts" appear next to the corresponding options. For a complete list, see Appendix G, "Accelerator Keys" on page 315 in *IBM Visualization Data Explorer User's Guide*.
2. The editor window can be invoked directly with the command:

```
dx -edit [program path name]
```

2.5 Controlling the Appearance of an Object: The Image Window

Controlling the appearance of an object means being able to control various aspects of its visual image on the screen as well as the data used in generating that image. This section deals briefly with control of the following aspects:

- the object's appearance (size and the "view" displayed in the Image window)
- the sequence in which object images are presented
- the input from which an object's image is generated
- the color(s) of an object
- the placement of axes around an object.

Note: Throughout this document, the term "Image window" refers to the window generated by Image (not by Display). Both of these modules are described in *IBM Visualization Data Explorer User's Reference*. Also see *IBM Visualization Data Explorer User's Guide* for a comprehensive treatment of the user interface.

Size, View, and AutoAxes

In Data Explorer the most easily controlled feature of an object in the Image window is its size, which can be changed by direct manipulation of the window. Other important features of its appearance are controlled through two options in the **Options** pull-down menu in the Image window menu bar: **View Control** and **AutoAxes**.

Size Control

To change the size of an image, simply change the size of the Image window. Holding down the *left* mouse button:

- Drag a horizontal border to shrink or expand the window vertically.
- Drag a vertical border to shrink or expand the window horizontally.
- Drag a corner to shrink or expand a window vertically *and* horizontally.

Notes:

1. **Reset** in the **Options** pull-down menu restores the original *view* of the object but not the original size of the window.
2. If your visual program uses the Display rather than the Image tool, the image size can be changed only by changing the resolution parameter of Camera or AutoCamera (see "Camera" on page 49 and "AutoCamera" on page 31 in *IBM Visualization Data Explorer User's Reference* for descriptions of these tools.)

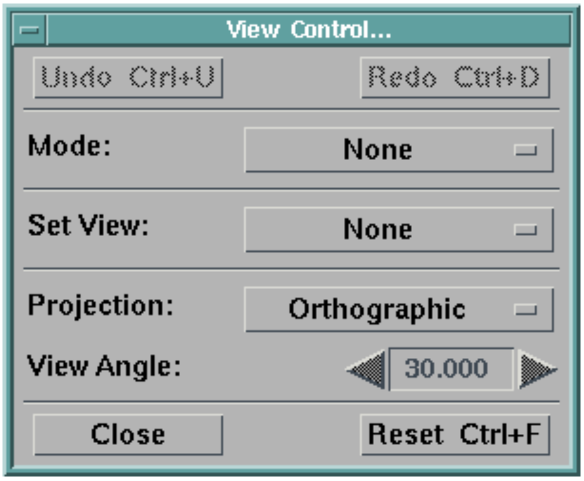
View Control

The **View Control** dialog box allows you to control (among other aspects of an object) the following:

- Viewing direction
- Rotation
- Field of view (zooming and panning).

To open this dialog box, select **View Control...** in the **Options** pull-down menu in the Image window.

Controlling the Viewing Direction: To change the viewing direction, select **Set View** in the dialog box (see Figure 6 on page 11). The list of choices that appears includes 7 directional views of the object (Top, Bottom, etc.). Because these "head



Mode: None

Set View: None

Projection: Orthographic

View Angle: 30.000

Figure 6. View Control Dialog Box. This dialog box is activated from the Options pull-down menu of the Image window (see Figure 5 on page 9).

on” views show only “one side” of an object (tending to flatten its appearance), the list also offers 7 corresponding “off” views (Off Top, Off Bottom, etc.).

When you select a view of an object, the image is automatically altered (note the highlighted **Execute** in the menu bar).

Controlling Rotation: To rotate an object in the Image window, first click on the **Mode** option box (initially displaying **None**) and then select **Rotate** from the displayed list. **Rotate** becomes the current mode and a set of axes appears in the lower right-hand corner of the window. You can rotate the object in two dimensions (clockwise and counterclockwise) or in three by rotating the axes. You can also cause the object to rotate “continuously” (i.e., in coordination with the axes).

2-D Rotation
: Position the mouse cursor in the Image window and hold down the *right* mouse button: clockwise movement of the mouse produces clockwise rotation of the *axes*; counterclockwise movement produces counterclockwise rotation. When the mouse button is released, the *object* rotates by the same amount as the axes have, assuming the same relative position in the window.

3-D Rotation
: Position the mouse cursor in the Image window and hold down the *left* mouse button: the mouse now behaves like a track ball and the axes move accordingly. When the mouse button is released, the object rotates by the same amount as the axes have, assuming the same relative position in the window.

Continuous Rotation
: Once you have selected **Rotate** mode, you can make the object rotate along with the axes: select **Execute** in the menu bar and then **Execute on Change** in the pull-down menu (**Execute** is highlighted). The image is now replaced by a dot representation of the object (see rendering options in *IBM Visualization Data Explorer User’s Guide*). The movement of the mouse (and the axes) is reflected directly and continuously in the movement of this dot version.

> **Note:** Movement of the axes is essentially synchronous with movement of the mouse, but the “response time” of the dot version may vary, depending on the machine and configuration.

Notes:

1. Turn off **Execute on Change** by selecting **End Execution** in the **Execute** pull-down menu.
2. To restore the original view of the object, select **Reset** in the **View Control** dialog box.

Controlling the Field of View: **Zoom** mode allows you to enlarge an object, making it appear closer (zooming in) or to reduce it, making it appear more distant (zooming out).

Pan/Zoom mode allows you to change the center of focus while zooming in or out.

Zooming in:

1. Select **Zoom** in the **Mode** option list of the dialog box.
2. To zoom in, position the mouse cursor in the Image window and hold down the *left* mouse button. An overlay rectangle appears.
3. You can enlarge or shrink this rectangle by moving the mouse cursor away from or toward the center of the window.
4. When you release the mouse button, the area of the rectangle expands to fill the Image window, making the object appear nearer.

   Note: If you simply click the mouse button instead of holding it down, the overlay rectangle will disappear before you can change its dimensions. The modified image of the object will be based on that rectangle. (The size of the rectangle and thus the degree of “zoom” depends on the distance of the cursor from the center of the window when you first press the mouse button.)
5. To cancel the effect of the most recent command, select **Undo** in the dialog box or in the **Options** pull-down menu. You can also repeat a command that has been “undone,” by selecting **Redo**.

   Note: Since executed commands are maintained in a stack, you can undo those commands one by one and redo them, too.
6. **Reset** in the dialog box (or in the **Options** pull-down) menu restores the original *view:* (e.g., front) of the object.

Zooming out:

1. Follow the procedure described for zooming in, but use the *right* mouse button.
2. When you release the mouse button, the area of the Image window is reduced to the area of the rectangle, making the object appear more “distant”.

Panning and Zooming out:

1. Select **Pan/Zoom** in the **Mode** option list of the dialog box.
2. Position the mouse cursor at the point in the Image window that you want as the center of the new “picture,” and press the appropriate mouse button (left to zoom in or right to zoom out).
3. Move the mouse in any direction to display the overlay rectangle. The “zooming” behavior of the object with respect to the rectangle will be the same as that just described.
4. To restore the original view of the object, select **Reset** in the dialog box.

5. Leave the **View Control** dialog box open for the next exercise.

AutoAxes Configuration

The **AutoAxes Configuration** dialog box allows you to generate a set of axes for an object in the Image window and to specify some of its characteristics.

1. Click on **Options** in the menu bar of the Image window and then select **AutoAxes** in the pull-down menu. The **AutoAxes Configuration** dialog box appears.

   Note: To display additional options, click on the **Expand** button at the bottom of the dialog box. For purposes of this tutorial, you will not be changing any **AutoAxes** options. For more information, see “AutoAxes” on page 27 in *IBM Visualization Data Explorer User’s Reference*.

2. Click on the **Enabled** toggle button (at the top of the dialog box) to select the AutoAxes option. The button is now activated.

3. Click on **OK** or **Apply** at the bottom of the dialog box to confirm the selection.

   Notes:

   a. **OK** closes the dialog box; **Apply** does not.
   b. Selection of **Enabled** and **OK** or **Apply** is necessary but not sufficient to activate the AutoAxes option (see next step).

4. Select **Execute** in the menu bar and then **Execute Once** in the pull-down menu.

   The object now appears in an axes box. Because the view is “head on,” the box appears to be 2-dimensional. For **Diagonal** or any of the “Off” views, however, it appears fully 3-dimensional (see next step).

5. To change the view of the object, select another (e.g., **Diagonal**) from the **Set View** pull-down list in the **View Control...** dialog box. The view changes to **Diagonal** (note that the axes box changes as well).

6. To remove the axes box from the window:

   a. Deactivate the **Enabled** toggle button in the **AutoAxes Configuration** dialog box by clicking on it.

   b. Click on **OK** or **Apply**.

   c. Again, execute the program (unless it is already in **Execute on Change** mode). The axes box disappears.

   d. To restore the original view of the object, click on **Reset** in the **View Control** dialog box

7. Close the **View Control** dialog box (by clicking on **Close**) and the **AutoAxes Configuration** dialog box (by clicking on **Cancel**).

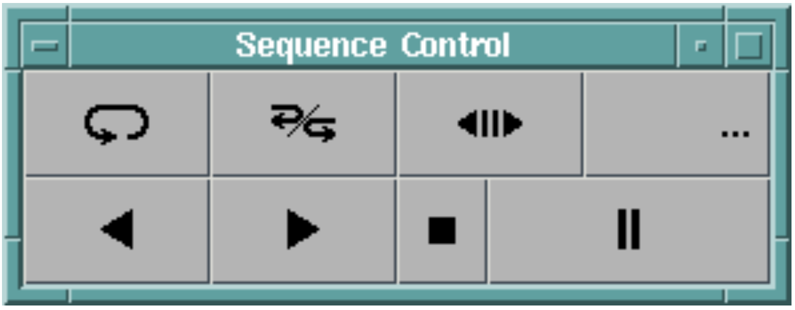


Figure 7. Sequence Control Panel. The first two buttons at top left are Loop and Palindrome. The others are: Step (◄||►), Counter (`...`), Back (◄), Forward (►), Stop (■), and Pause (||).

Using the Sequencer

The Sequencer allows you to "animate" a visual image and is very easy to use. The process is rather like running a video cassette tape: You can play it forward or backward, stop it, pause, and so on. (If you look at the canvas of the VPE window, you will see that the Sequencer is one of the components of the `example1.net` network.)

1. Click on **Execute** in the menu bar of the Image window and then on **Sequencer** in the pull-down menu to display the Sequence Control panel (Figure 7).
2. Click on the Forward button (►) to start the animation sequence.

   Note: A Sequencer button appears recessed when it is activated.

   As the sequence proceeds, the Counter button (`...`) displays the corresponding "frame" number. At the end of the execution cycle, the window displays the final image in the sequence, the Counter displays the final frame number, and the Forward button is deactivated (not recessed).
3. Click on the Back button (◄) to run the same sequence in reverse. At the end of the execution cycle, the window displays the original image, the Counter displays "0", and the Back button is deactivated.
4. Click on the Step button (◄||►) to activate it. You can now proceed through a sequence frame by frame in either direction, using Back and Forward.

   Note: In Step mode, the Back and Forward buttons appear as ◄|| and ||► respectively.
5. Click on Step again (to deactivate it) and then on the Loop button at the top left (marked by a single "looped" arrow). Now, if you click on Back or Forward, the sequence will repeat itself until you interrupt it (see the next four steps).
6. Click on the Pause button (||) to suspend the sequence.
7. Continue the sequence from this point or reverse it, using Back or Forward.
8. Click on the Stop button (■). The sequence halts (the Loop button remains activated).

   Note: The Stop button does not affect the status of the Loop, Palindrome, or Step button.
9. Click on Back or Forward. A new sequence starts again from the beginning (i.e., a "stopped" sequence cannot be continued or reversed from the point at which you interrupted it).

10. Click on Loop. The sequence continues to its end before stopping and the Loop button is deactivated (compare with the Stop button).
11. Click on the Palindrome button (between Loop and Step). With this option activated, you can use the Back (or Forward) button to run a sequence through one back-and-forth cycle (from first frame to last and back to first, or vice versa). Note that if you activate this function at some intermediate frame in the sequence, only the *remainder* of the cycle is executed.
12. To restore the original view of the object, click on **Reset** in the **View Control** dialog box.
13. Close the Sequence Control panel (double click on the window menu button in top left corner of the frame).

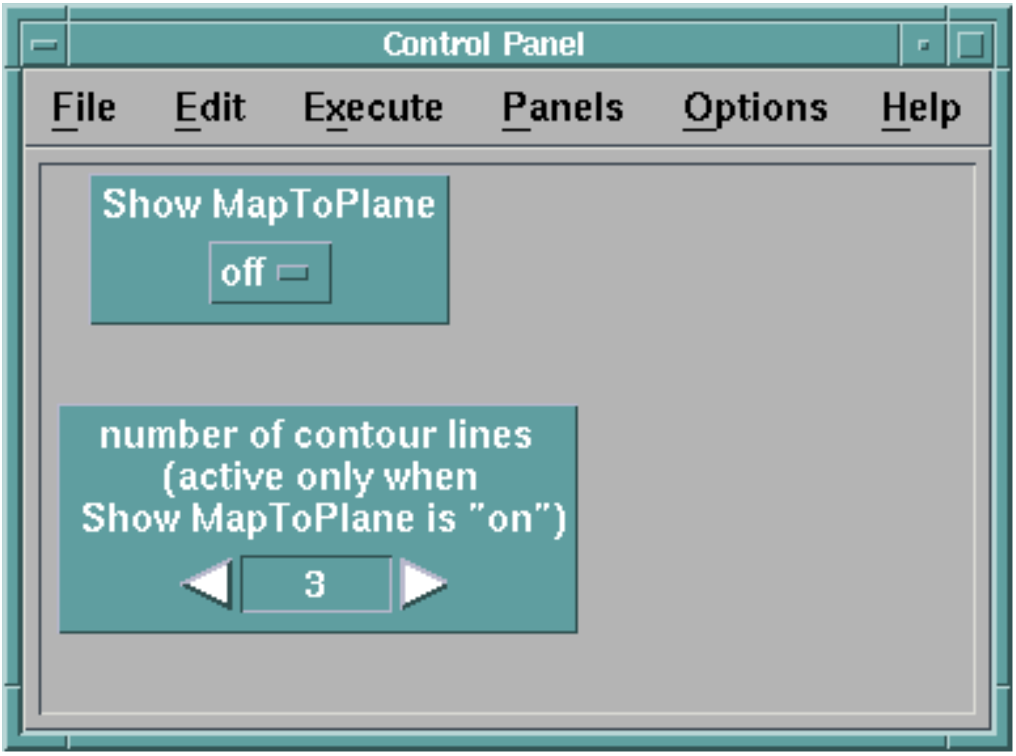
For Future Reference

You can activate both the Loop and Palindrome buttons together. The back-and-forth cycle will repeat itself until you deactivate one them or click on the Stop button.

Using Control Panels

Control panels give you direct control of inputs to a visual program. The control panel included with `example1.net`, for example, allows you to incorporate a colored plane in the image of an object and to decide the number of contour lines to be displayed in that plane. To open the control panel:

1. Select **Windows** in the menu bar of the Image window.
2. Select **Open All Control Panels** in the pull-down menu. The control panel appears (see Figure 8).



Show MapToPlane
off

number of contour lines
(active only when
Show MapToPlane is "on")
3

Figure 8. Control Panel with two Interactors..

To display a colored plane:

1. Click on **`off`** in the Show MapToPlane interactor.
2. Select **`on`** when it appears in the selection list.
3. Select **`Execute`** in the menu bar of either the control panel or the Image window.
4. Select **`Execute on Change`** in the pull-down menu. The visual program reexecutes and the colored plane is incorporated as part of the current image.
5. To specify the number of contour lines, click on one of the two stepper arrowheads in the **`number of contour lines`** interactor (right to increase the number, left to decrease it). Since Data Explorer is in **`Execute on Change`** mode, the number of contours changes when the number in the interactor changes.

   Notes:

   a. The second interactor has no effect if the *first* interactor is **`off`**.

   b. Depending on opacity and other factors, some of the contour lines "inside" the object may not be visible.
6. Click on **`on`** in the Show MapToPlane interactor and then select **`off`** when it appears. The plane disappears and the original image is restored.

Using the Colormap Editor

The `example1.net` visual program includes a Colormap Editor for controlling the color characteristics of data values represented in the visual image. The editor also controls the *opacity* of those values (see Table 1 on page 17).

Opening the Colormap Editor:

1. Select **`Open All Colormap Editors`** in the **`Windows`** pull-down menu of the Image window. Now changes made in the Colormap Editor (Figure 9 on page 18) will be reflected both in the editor and in the image. (Data Explorer should still be in **`Execute on Change`** mode from the preceding exercise. If not, select that mode in any **`Execute`** pull-down menu.) For example:
2. Move the mouse cursor into the large rectangle under the **`Hue`** label button (the cursor changes from an arrow to a circle with sight marks).
3. Use the left mouse button to drag the *control point* (the small box in the upper left-hand corner of the rectangle) horizontally to a position under **`Hue`**. The color column to the left changes as the point moves.

   Note: The **`Hue`** button is automatically activated when the mouse button is pressed with the cursor positioned in the **`Hue`** rectangle.
4. Release the mouse button. Color changes appear in the object.
5. Drag the control point back to its original position and release the mouse button. The color column and the object return to their original states.

Characteristic	Description	Numerical Range/ Image Values
Hue	A particular color	0.000 = Red 0.333 = Green 0.666 = Blue 1.000 = Red
Saturation	Purity of color	0.0 = White (all colors) 0.5 = “Pastel” (50% of one color) 1.0 = Pure (100% of one color)
Value	Degree of brightness	0.0 = Black (0% brightness) 0.5 = Dark (50% brightness) 1.0 = Maximum (100% brightness)
Opacity	Degree of transparency	0.0 = 0% Opaque (100% transparent) 0.5 = Semi-opaque (50% transparent) 1.0 = 100% Opaque (0% transparent)

Table 1. Image Characteristics Controlled by the Colormap Editor. The numerical range used in specifying discrete values of a characteristic is 0.0–1.0. Corresponding Colormap values at the limits and middle of a range are also listed.

Before going any further, you should familiarize yourself briefly with the Colormap Editor (Figure 9 on page 18). Note that the four labeled rectangles to the right correspond to the characteristics listed in Table 1. In each rectangle, at least two control points (very small “boxes”) and a line connecting them determine how the image characteristics associated with a particular data value are represented in the image (the figure shows the default settings).

Changes in the position of a control point are reflected directly in the two left-hand columns of the editor and (in **Execute on Change** mode) in the Image window. The “data range” from the bottom to the top of the color column is the range of actual data values for which image characteristics (e.g., hue) can be specified.

Specifying Colormap Values

You can specify a Colormap value in two ways:

- Directly by manipulating a control point.
- Indirectly through the **Add Control Points** dialog box.

The first is quick and approximate. The second is slower and precise.

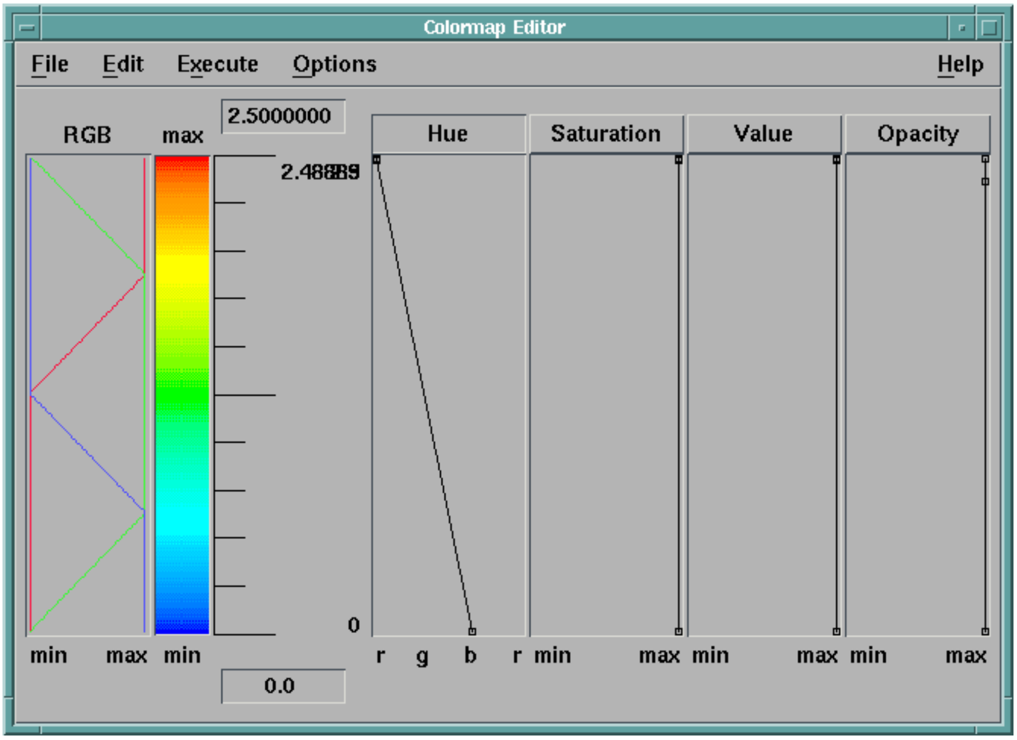


Figure 9. Colormap Editor for .../example1.net. The four image characteristics controlled by the editor (hue, saturation, value, and opacity) are also listed in Table 1. The numerical range for each is 0.0–1.0. For **`Hue`** *this range is "r g b r" (red green blue red); the current setting corresponds to the red-green-blue spectrum of the color column to the left. The column furthest left (RGB) displays the red-green-blue color values corresponding to the four settings on the right. Note that each setting is determined by a "line" whose position can be changed by moving the "control point" at one of its ends; the shape of the line can be modified by adding control points (see text). The values 0.0 and 2.5000000 are the minimum and maximum of the data values used in the visual program.*

`Specifying approximate values:`

1. Position the mouse cursor in the middle of the large **`Value`** rectangle and *double click* the left mouse button. Note the changes that result:
    - A new control point appears (turning the vertical line into two line segments) and the **`Value`** button is activated.
    - The data value of the new point is displayed in the data range next to the color column (the RGB area, color column, and image show corresponding changes).
2. To move the control point, use the left mouse button to drag it. The RGB area, color column, data value, and image all change accordingly.
3. To return Value to its previous state, double click on the control point.

`Specifying exact values:`

1. Select **`Edit`** in the menu bar of the Colormap Editor and then select **`Add Control Points`** in the pull-down menu. A dialog box appears (see Figure 10 on page 19).

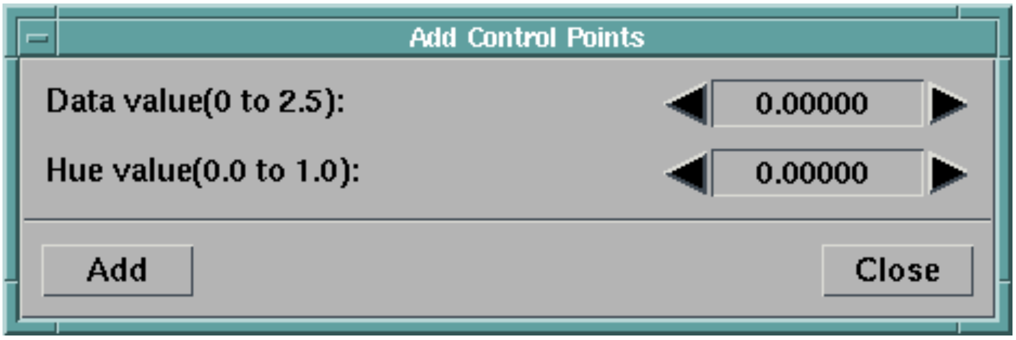


Figure 10. Add Control Points Dialog Box. This dialog box has two interactors: The first specifies the actual data value for which a corresponding set of Colormap values are to be implemented. The second specifies one of the four Colormap values, such as **`Hue`** *in this example (see text).*

2. Click on the **`Saturation`** rectangle of the Colormap Editor. The label button in the editor is activated (recessed) and the new interactor label in the **`Add Control Points`** dialog box changes to “Saturation value (0.0 to 1.0).”
3. Click on the data value displayed in the dialog box, type in the value `1`, and press **`Enter`**. The new value is displayed as `1.00000`.
4. Click on the saturation value displayed in the dialog box, type in the value `.28`, and press **`Enter`**. The new value is displayed as `0.28000`.
5. Click on **`Add`** in the dialog box: A control point corresponding to the new values in the dialog box appears in the **`Saturation`** rectangle of the editor. The RGB area, color column, and image all change accordingly.

Now that you know how to open and execute visual programs and to control the images they generate, you can proceed to Tutorial II, which deals with various techniques for visualizing data.

Additional Notes on Control Points

Displaying Control Point Values

By default the Colormap Editor displays the values of all control points. To display the value of an individual point:

1. Select in order:
   a. **Options** in the menu bar of the ColorMap Editor
   b. **Display Control Point Data Value** in the pull-down menu
   c. **Selected** in the pull-down list.
2. Click on the desired control point in the ColorMap Editor (the rectangle containing the control point is automatically activated if it is not already activated). The data value corresponding to the selected control point appears in the data range next to the color column.
3. Repeat the preceding selection procedure, except click on **All** in the pull-down list (instead of **Selected**). Data values for all control points in the *activated* rectangle now appear in the data range.

Deleting Control Points

You can delete control points one at a time or in groups:

- To remove control points one at a time (in an activated rectangle), *double click* on each point to be removed
- To remove two or more points at a time:
  1. Select points by either:
     – Shift-clicking: Press the Shift key and click on each point to be deleted; *or*
     – Drawing a selection box: Hold down the left mouse button and drag the cursor to generate a “selection box” in the selected rectangle and enlarge it to the desired size. Release the button (the box disappears, but any “boxed” points are selected).
  2. Select **Edit** in the menu bar and then **Delete Selected Control Points** in the pull-down menu. All selected control points are deleted and the image is updated.

Chapter 3. Tutorial II: Editing and Creating Visual Programs

3.1 Editing a Visual Program: The Basics 22
3.2 Creating a Visual Program: Two short examples 24
  A simple two-dimensional field 24
  A simple three-dimensional field 24
3.3 Importing Data 25
  Example 1 25
  Example 2 26
3.4 A thumbnail Sketch of the Data Prompter Choices 27
3.5 Importing Your Own Data 28
3.6 Visualizing 2-Dimensional Data 29
  Colors 29
  Contour Lines 29
  Streamlines 30
  RubberSheet 30
  2-D Scalar Glyphs 31
  2-D Vector Glyphs 32
3.7 Visualizing 3-Dimensional Data 32
  Isosurfaces 32
  Slices 32
  Streamlines 33
  3-D Scalar Glyphs 34
  3-D Vector Glyphs 34
  Volume Rendering 34
3.8 Tasks and Tools 36
  Adding Captions 36
  Adding Input Tabs to Tool Icons 37
  Connecting Scattered Data Points 37
  Controlling Execution with Switch 37
  Controlling Inputs: Configuration Dialog Boxes 38
  Controlling Inputs: Interactors 39
  Creating Animations 41
  Creating and Using Macros 42
  Data-driven Tools 43
  Modules: Using AutoColor 44
  Modules: Using Compute 44
  Modules: Using Map 45
  Modules: Using Plot 46
  Processing Images 46
  Saving and Printing Images 47
3.9 Scripting Language 47

© Copyright IBM Corp. 1991-1997

This tutorial will show you how to modify existing visual programs and create new ones. In the process, it will also introduce you to a number of the most commonly used modules. As you become more experienced using Data Explorer, you can explore more of each module's many options.

Tutorial II Visual Programs

- Visual programs and files required for Tutorial II all reside in the same directory and therefore have the same path name except for file name and extension. Thus where a visual program is referred to only by its file name and extension (i.e., `.../filename.ext`), the full path name is easily derived if needed:

  `/usr/lpp/dx/samples/tutorial/filename.ext`

- The procedures you will be using most are:
  - 2.4, "Opening and Executing a Visual Program" on page 6
  - selecting tools and placing their icons in the Visual Program Editor canvas (see "Selecting tools and placing icons")
  - opening and modifying a configuration dialog box (see "Specifying inputs: configuration dialog boxes" on page 23 and also "Controlling Inputs: Configuration Dialog Boxes" on page 38).

3.1 Editing a Visual Program: The Basics

When you open the Visual Program Editor (VPE) with the `dx -edit` command, or by choosing **New Visual Program** in the Startup window, you will see a large blank area (the "canvas") and two "palettes" to its left (see Figure 4 on page 7). The palette at top lists "categories" of tools (modules). The palette below it lists the tools in the currently selected (highlighted) category.

Selecting tools and placing icons

1. Click on a tool category (e.g., **Transformation**) in the upper palette. The tools in that category are now listed in the lower palette.
2. Click on a tool in the lower palette (e.g., **AutoColor**) to highlight it.
3. Move the cursor into the canvas area. Note that it becomes an inverted "L."
4. Position the cursor at the point where you want the tool icon to appear and click again to generate the icon.

Tool icons

Each tool icon has one or more tabs on top and bottom. These tabs represent, respectively, input(s) to the tool module and output(s) from it.

There are two ways to specify an input:

- by specifying a value in the configuration dialog box associated with a particular tool
- with an arc, or line, connecting the output tab of one icon to the input tab of another.

Specifying inputs: configuration dialog boxes

To open the configuration dialog box for a tool, either:

- Double click on the tool icon in the canvas, *or*
- Single click on (highlight) the tool icon in the canvas and select "Configuration" from the `Edit` pull-down menu.

In the case of interactors, color maps, the Image tool, and the Sequencer, the first procedure invokes a "control box" (another kind of dialog box). The second procedure always invokes a configuration dialog box.

Tabs and inputs to a tool module

Each input parameter in the configuration dialog box corresponds to an upper tab on the tool icon. The leftmost tab corresponds to the first parameter, and so on.

Visible versus hidden parameters

At the bottom of the dialog box are `Expand` and `Collapse` buttons. The first button "expands" the dialog box, displaying additional parameters. The second button "collapses" the dialog box, hiding every parameter whose "Hide" toggle button is activated. (When a tool has no hidden parameters, both buttons are disabled, as indicated by their gray labels.)

Specifying inputs: arcs and icons

A visual program is a "network" of interconnected tool modules. In the VPE window, this network is represented as a set of tool icons connected to one another by lines ("arcs") representing the data (Figure 4 on page 7 in Tutorial I shows such a network.)

Connecting tool icons with arcs

1. Position the cursor on an output tab of an icon and hold down the left mouse button: the cursor becomes a downward-pointing arrow, and a parameter or data name for that tab appears on the icon.
2. Drag the cursor to another icon: the output tab remains connected to the cursor by a highlighted line (arc). If the connection is valid, the input tab(s) will be highlighted when the cursor arrow touches the icon. (If the connection is invalid, the tab(s) will not be highlighted, and an error message will appear.
3. Release the mouse button to establish a connection to a valid input tab.

Notes:

a. If the "receiving" icon has two or more valid input tabs, if you release the mouse button when the cursor is on the main part of the icon, the connection to the first (leftmost) tab is automatic. To establish a connection to a different tab requires placing the cursor on that tab before releasing the mouse button.
b. You can establish a connection starting with an input tab and dragging the arc to an output tab.

Disconnecting or moving an arc

Click on the input tab to which the arc is connected (e.g., the input tab on Sequencer): a highlighted arc, connected to an (Import) output tab appears:

- Disconnect the arc by dragging the cursor to an empty spot in the canvas and releasing the mouse button, *or*
- Move the arc by dragging it to another icon and releasing the mouse button to establish the new connection.

Deleting a tool icon

1. Highlight the tool icon by clicking on it.
2. Select **`Delete`** in the **`Edit`** pull-down menu.   .

Moving a tool icon

Select a tool icon and drag it to the desired location before releasing the mouse button.

3.2 Creating a Visual Program: Two short examples

If necessary, review 3.1, "Editing a Visual Program: The Basics" on page 22.

A simple two-dimensional field

In this first example, we will import a two-dimensional field on a regular grid. The data describe the topography (elevation) of the southeastern United States. First start Data Explorer by typing

```
dx -edit
```

or choose **`New Visual Program`** from the Data Explorer Startup window.

From the Import and Export Category in the top left palette, choose the Import module. Place it on the large blank canvas area.

From the Realization Category choose the RubberSheet module. Place it below the Import module. From the Rendering Category choose the Image module. Place it below the RubberSheet module.

Now connect the output of Import to the first input of RubberSheet, and connect the output of Rubbersheet to the input of Image. Notice that the first tab of Import is colored differently than any other tabs visible. This is because this parameter is *required*. This is the file name of the data file to import.

Open the configuration dialog box for Import. Specify the first parameter (name) as `/usr/lpp/dx/samples/data/southeastern_topo.dx`.

Execute the visual program by selecting **`Execute Once`** from the **`Execute`** menu. You should see an image showing the topography as a deformed surface.

To add colors, insert an AutoColor module (Transformation Category) between RubberSheet and Image (or between Import and RubberSheet).

A simple three-dimensional field

In this second example, we will import a three-dimensional field on a regular grid. The data describe the cloud water density in a simulation of a storm.

As before, place an Import tool from the Import and Export Category on the canvas. Place an Isosurface tool from the Realization Category below Import. Place an Image tool from the Rendering Category below Isosurface. Connect the output of Import to the first input of Isosurface, and connect the output of Isosurface to the input of Image.

Open the configuration dialog box for Import, and type in the **`name`** parameter as `/usr/lpp/dx/samples/data/cloudwater`.

Execute. You will see an isosurface (constant value surface) of the cloudwater density. Not that if you are using the same Image tool as in the previous example, you may need to "reset the camera" for the new data set. Choose **Reset** from the **Options** menu of the Image window.

Open the configuration dialog box for Isosurface. Notice that the default for the isovalue (the second input) is the data mean. Change it to the value 0.1 by typing over the **Value** field. Execute.

For more complex visualizations, continue with the examples in 3.6, "Visualizing 2-Dimensional Data" on page 29, 3.7, "Visualizing 3-Dimensional Data" on page 32, and 3.8, "Tasks and Tools" on page 36. A short tutorial on importing data may be found in 3.3, "Importing Data" and 3.5, "Importing Your Own Data" on page 28.

3.3 Importing Data

Of the data formats that Data Explorer can import, the General Array format is likely to be the most useful to a majority of users. Other formats are discussed in detail in Appendix B, "Importing Data: File Formats" on page 241 in *IBM Visualization Data Explorer User's Guide*.

The General Array format uses a simple header file to describe data characteristics such as grid dimensions, data type, and layout. The two examples here use the Data Explorer Data Prompter to create such header files for two simple data files. The Data Prompter is designed specifically for importing data in General Array format. (For details of format and the Data Prompter, see Chapter 5, "Importing Data" on page 61.)

Example 1

This example illustrates the importation of a simple data set consisting of a single variable on a 5 × 5 × 5 grid.

1. First enter:

   ```
   dx -prompter
   ```

   You can also access the Data Prompter by choosing **Import Data** from the Data Explorer Startup window.

   When the initial dialog box appears (see Figure 15 on page 99), choose Grid or Scattered File, and then type the path name `/usr/lpp/dx/samples/tutorial/external.data`. into the **Data file name** field at the top of the dialog.
2. Press the **Describe Data** button to bring up a window which allows you to describe the data.
3. You can now view the file by clicking on the ellipsis button (**...**) to the right of the file-name field and selecting **Browser** from the pull-down menu. The **File Browser** window appears. (If necessary, move this window so that you can view it and the Data Prompter window at the same time.) It is also possible to browse the data file from the initial prompter window once the file name has been entered.
4. The first three lines of the file may look like data, but they are header information, so click on the **Header** toggle button and then on **# of bytes** to the right (now activated).

5. Select **# of lines** and enter the value “3” in the associated field.
6. The first line of the header gives the grid dimensions as `5 5 5`, so enter these numbers in the first three **Grid size** fields. Note what happens in the **Grid positions** fields as each number is entered. (You can use the tab key or the mouse to move the cursor from one field to the next.)
7. For **Data format**, ensure that **ASCII (text)** is selected.
8. Next you need to specify whether the data is in row or column majority order. In this particular file, the data is in row majority, so select that button.
9. Position the cursor in the first **origin, delta** field, hold down the left mouse button, and drag the cursor over the numbers there to highlight the field before releasing the mouse button. The default values and the highlight bar will disappear as soon as you start typing.
10. The origin of the grid is [1.0, 3.0, 2.0] (second line of the header) and the corresponding deltas are .5, .3, and .8, respectively (third line). So enter the values:

    ```
    1, .5
    3, .3
    2, .8
    ```

    in the three **origin, delta** fields.
11. On the right side of the prompter window, you can change the name of the data variable (`field0` by default) and specify the data type and structure. For this example, the data are floating-point scalar, so you do not need to change the settings.
12. You can now save the header file you have defined. Select **Save As...** from the **File** pull-down menu. Save the file under any name you choose.

    Note: The data can now be imported by specifying this file name to an **Import** module. However, if the extension is not “general,” you must specify “general” in the “format” parameter field of the **Import** configuration dialog box (see “Controlling Inputs: Configuration Dialog Boxes” on page 38).
13. In the initial Data Prompter window, note that the **Test Import** and **Visualize Data** buttons are now enabled. First choose **Test Import**. A window appears, displaying a description of the imported data. Now choose **Test Import**. A general purpose visualization program will be run on this data set. To view the program, choose Open Visual Program Editor from the Windows menu of the Image window. You can also experiment with changing the interactor settings in the control panel.

Example 2

In this example, you will use the initial dialog box to customize the Data Prompter before importing a data file that contains scattered data values for two variables. The organization of the file is:

```
x, y, data1, data2
x, y, data1, data2
. . .
```

where `x, y` define the positions (or locations) of the data. (See also “For Future Reference” later in this example.)

1. First, invoke the Data Prompter (as in Example 1, Step 1). When the initial dialog box appears, click on the **Grid or Scattered File** button for scattered data. Then click on the **Grid Type** button for scattered data (farthest right).
2. Use a stepper button to set **Number of variables** to “2.”

3. The data positions are specified in the data file itself, so activate the **`Positions in data file`** toggle.
4. The data positions are 2-dimensional (x,y), so use a stepper button to set **`Dimension`** to “2.”
5. Verify that the **`Single time step`** toggle is activated.
6. Set **`Data Organization`** to **`Columnar`**.
7. Click on **`Describe Data`**. The simplified prompter window appears.
8. Position the mouse cursor in the **`Data file`** field at the top left and type in the path name `/usr/lpp/dx/samples/tutorial/spreadsheet.data`.

   You can view the file by choosing **`Browser`** (as in Example 1).
9. There is no header in this file. Set **`# of Points`** to “49.”
10. To save the header file you have defined, select **`Save As...`** in the **`File`** pull-down menu. Save the file under any name you choose.
11. The **`Test Import`** and **`Visualize Data`** buttons in the initial Data Prompter window are now enabled. Choose **`Visualize Data`** to see a visualization of this scattered data.

Many other examples of using the General Array format can be found in 5.1, “General Array Importer” on page 63. Data Prompter options are described in 5.4, “Data Prompter” on page 96.

For Future Reference

It is important to note that the top-to-bottom order of items in the **`Field list`** (right side of window) is the same as the left-to-right order of items in the data file itself:

`locations`	`x, y`
`Field0`	`data1`
`Field1`	`data2`
...	...

(“locations” is a General Array reserved word used to indicate when numbers in a data file are to be interpreted as “positions”).

The order of the field list can be changed (with the **`Move field`** stepper buttons) to agree with the order in the data file.

If other information (e.g., descriptive text) is interspersed among the data values, you must use the layout options available in the full Data Prompter (see Layout on page 108 and “layout” on page 92).

3.4 A thumbnail Sketch of the Data Prompter Choices

Data Explorer File

This choice is for when you already have data stored in Data Explorer format, for example if you are using a filter which converts from another format to Data Explorer format.

CDF Format

This is a standard data format which Data Explorer imports directly. An application programming interface can be obtained from the National Space Science Data Center, NASA Goddard Space Flight Center.

netCDF Format

This is a standard data format which Data Explorer imports directly. An application programming interface for netCDF can be obtained from the Unidata Program Center in Boulder, Colorado.

HDF Format

This is a standard data format which Data Explorer imports directly. HDF was created at the National Center for Supercomputing Applications.

Image File

Data Explorer directly imports TIFF, MIFF, GIF, and RGB format images.

Grid or Scattered File

This option allows you to specify a general array header which can describe a wide variety of formats of data. Data can be on a regular grid, a warped grid with positions explicitly specified, or scattered. The general array format can skip header lines and embedded descriptive text. Data can be multidimensional, multivariable, and series (see 5.1, "General Array Importer" on page 63).

Spreadsheet Format file

This option is typically used for non-spatial data, for example the output of a spreadsheet program which might look like:

```
Customer       Sales        Profit
company_a       1.2           938
company_b       8.9          1037
    .            .             .
    .            .             .
    .            .             .
```

See "ImportSpreadsheet" on page 170 in *IBM Visualization Data Explorer User's Reference*.

3.5 Importing Your Own Data

The Data Prompter gives you access to some general purpose visualization programs which can visualize a wide variety of data types. The intent is to get you a picture of your data as quickly as possible. So start by importing your data through the Data Prompter and then choose the Visualize Data button. You can see the visual program used by choosing Open Visual Program Editor from the Windows menu of the Image window. You can save the visual program by choosing the **Save As** option from the Visual Program Editor **File** menu.

The following sections in this tutorial include a number of examples of different types of visualizations that can be performed on data. Each one uses sample data provided with Data Explorer. If you want to apply a particular visualization method (program) to your own data, proceed as follows:

1. Create a General Array Format header file for your data with the Data Prompter (see 3.3, "Importing Data" on page 25 and 5.4, "Data Prompter" on page 96) or use any other available method of importation (see *IBM Visualization Data Explorer User's Guide*).
2. Starting with the visual program you want to use, open the **Import** configuration dialog box and change the **name** parameter to the name of the file to be imported (either the General Array header file or the data file if another format is used).

3. Execute the visual program using your data file.

   Note: You will probably have to "reset the camera" to see your data, because the program you choose has viewing parameters appropriate to the sample data but not necessarily to yours. To reset the camera, select **`View Control`** in the **`Options`** pull-down menu of the Image window. Click on **`Reset`** at the bottom of the dialog box.

3.6 Visualizing 2-Dimensional Data

This section describes several ways of visualizing 2-dimensional scalar and vector data.

Colors

Using the AutoColor tool to color 2-dimensional data is the simplest visualization method available. By default, AutoColor assigns blue to the smallest values and red to the largest. (If the data are vector, the colors are based on the magnitude of the vectors).

1. Open and execute visual program `.../AutoColor2D.net`.

   The resulting image is a map of temperatures around the world: highest near the equator and over continents; lowest near the poles and in the oceans.
2. To display the temperature values associated with each color, connect the output tab of the **`ColorBar`** icon in the VPE window to the open input tab on the **`Collect`** icon.
3. Reexecute the visual program.

The **`Color`** tool and the **`Colormap Editor`** provide additional control of the color map:

1. Open and execute visual program `.../Color2D.net`.
2. Open the **`Colormap Editor`** by double clicking on the **`Colormap`** icon.
3. Select **`Execute on Change`** in the **`Execute`** pull-down menu. Any change you now make in the **`Colormap Editor`** appears immediately in the image. For example, you can experiment with:
   - moving control points (clicking on and dragging)
   - adding control points (double clicking at the desired location)
   - deleting control points (double clicking on individual points).
4. Connect the output tab of the **`ColorBar`** icon in the VPE window to the open input tab on the **`Collect`** icon.
5. Select **`Execute on Change`** in the **`Execute`** pull-down menu. The color bar from the **`Colormap Editor`** now also appears in the image. Again, a change made in the editor is immediately reflected in the image.

For more information, see AutoColor, Color, and ColorBar, in *IBM Visualization Data Explorer User's Reference*.

Contour Lines

Contour lines connect points of the same value in a 2-dimensional data set. The visual program in this example uses elevation data for the southeastern United States.

1. Open and execute visual program `.../Isosurface2D.net`.

   The contour line that appears in the image has an isosurface value of 0 meters (sea level).

2. Open the **Isosurface** configuration dialog box.
3. In the **value** parameter field, change “0.0” to “20” and click on **OK**. The dialog box closes.
4. Select **Execute on Change** in the **Execute** pull-down menu and note the change in the contour line.
5. Reopen the Isosurface configuration dialog box and type “0 20” in the **value** parameter field.
6. Click on **OK**. The image now consists of two contour lines.

For more information, see Isosurface in *IBM Visualization Data Explorer User’s Reference*.

Streamlines

If your 2-dimensional data set consists of vectors, you can create streamlines that trace the path of a massless particle in a vector field.

1. Open and execute visual program `.../Streamlines2D.net`. The image is a set of streamlines that follow a wind field over the surface of the earth.
2. Change the number of streamlines by changing the value of the **density** parameter in the configuration dialog box of the **Sample** tool.
3. Reexecute the visual program.

Note: Streamlines can be generated in other ways as well. For example:

1. Pass a list of 2-dimensional positions to the **start** parameter in the **Streamline** configuration dialog box by either:
   a. inserting a **VectorList** interactor stand-in (from **Interactor** in the categories palette) into the visual program, and connecting it to the **start** parameter tab of Streamline *or*
   b. typing the start positions in the configuration dialog box for **Streamline**.

2. Use the **Grid** tool (from **Realization** in the categories palette) to create a particular set of start positions. Any field that contains positions can be used as the starting point for streamlines.

   This sample visual program contains a Grid tool. Connect it to the **start** parameter tab of Streamline. It generates a 3 × 3 grid of points. Modify it to create a 10 × 3 grid of starting points.
3. Use the **Probe** tool (from **Special** in the categories palette) to select starting points in the Image window.

   This sample visual program contains a Probe tool. (Compute is used to make the value of the probe point 2-dimensional.) Connect the output of Compute to the **start** input tab of Streamline, and then move the probe (i.e., enter cursor mode, using the **View Control** dialog box).

See Grid, Probe, and Streamline in *IBM Visualization Data Explorer User’s Reference*.

RubberSheet

You can “warp” the representation of 2-dimensional data with the **RubberSheet** tool.

1. Open and execute visual program `.../RubberSheet.net`. The image represents elevation above sea level in the southeastern United States. Regions of greatest elevation are colored red; regions of lowest elevation, blue.
2. Open the **RubberSheet** configuration dialog box.

3. Change the value in the **scale** parameter field to ".002."
4. Click on **OK** and reexecute the visual program. The contrast of variation is greatly increased.

For more information, see RubberSheet in *IBM Visualization Data Explorer User's Reference*.

You can also add contour lines to a Rubbersheet image:

1. Select **Realization** in the categories palette and then **Isosurface** in the tools palette.
2. Place the icon to the right of **RubberSheet**.
3. Connect the output of **RubberSheet** to the first input tab of **Isosurface** (the only connection that Data Explorer will allow you to make). Make sure that you do not break the connection to **Shade**.
4. Open the **Isosurface** configuration dialog box.
5. Change the **number** parameter value to "10" and click on **OK**.
6. Select **Structuring** in the categories palette and **Collect** in the tools palette.
7. Place the **Collect** icon below **Isosurface**.
8. Connect the outputs of both **Shade** and **Isosurface** to the inputs of **Collect** (again, Data Explorer will allow only valid connections).
9. Disconnect **Image** from **Shade**.
10. Connect the output of **Collect** to the input of **Image**.
11. Reexecute the visual program. The result is a set of colored contour lines combined with the Rubbersheet image.

You can change the color of the contour lines:

1. Select **Transformation** in the categories palette and then **Color** in the tools palette.
2. Place the **Color** icon to the right of **RubberSheet** and **Collect**.
3. Open the **Color** configuration dialog box.
4. Delete "(no color added)" in the **color** parameter field and type in "black" (the quotation marks are unnecessary and will be added by Data Explorer).
5. Click on **OK**.
6. Move the output of **Isosurface** from **Collect** to **Color**.
7. Connect the output of **Color** to the input of **Collect**.
8. Reexecute the visual program.

See Color and Isosurface in *IBM Visualization Data Explorer User's Reference*.

2-D Scalar Glyphs

In this example, elevation data are represented both by color and by glyphs (the black circles).

1. Open and execute visual program `.../AutoGlyph2DScalar.net`. The elevation values range from higher (orange; larger circles) to lower (blue; smaller circles). The size of each glyph (circle) is proportional to the data value it represents. To display these values:
2. Open the configuration dialog box for **AutoGlyph**.
3. In the **type** parameter field, replace "(input dependent)" with "text" and click on **OK**.
4. Reexecute the program. The actual data values that appear are called *text* glyphs.

See AutoGlyph in *IBM Visualization Data Explorer User's Reference*.

2-D Vector Glyphs

Glyphs can be used to represent vector as well as scalar data.

- Open and execute visual program `.../AutoGlyph2DVector.net`.

  Again, data values are represented by color and by glyphs (arrows). In this example of wind velocity data, colors represent the magnitude and arrows the direction. (The black squares signify missing data or data omitted as invalid by the **Include** module in the program.)

See AutoGlyph in *IBM Visualization Data Explorer User's Reference*.

3.7 Visualizing 3-Dimensional Data

The following examples illustrate several ways of visualizing 3-dimensional data.

Isosurfaces

This example uses cloud-water density data for a severe storm.

1. Open and execute visual program `.../Isosurface3D.net`.

   The "isovalue" used for generating the isosurface that appears is, by default, the average of all the data values. (The default isovalue can be found by selecting **Open Message Window** in the **Windows** pull-down menu.)
2. Change the isovalue:
   a. Open the configuration dialog box for **Isosurface**.
   b. In the **value** parameter field, set the value to "0.3."
   c. Click on **OK** and reexecute the visual program. The new isosurface is significantly smaller.

See Isosurface in *IBM Visualization Data Explorer User's Reference*.

Slices

Following are a few examples of how to generate and process data slices.

For 3-dimensional data on any type of grid or for non-orthogonal slices, use the **MapToPlane** tool. If the data set is on a regular grid, use the **Slab** tool to take slices along connection elements. You can use other tools (e.g., **AutoColor** or **RubberSheet**) on data slices just as you can on any 2-dimensional object.

1. Open and execute visual program `.../MapToPlane.net`. The image is a colored plane through a 3-dimensional data field. By default, **MapToPlane** maps onto a plane at the center of the data. To change the orientation of this plane:
2. Open the configuration dialog box for **MapToPlane**, change the value in the **normal** parameter field to [1 1 1] and click on **OK**. The change of orientation will appear when the program is reexecuted.
3. Reexecute the visual program. **MapToPlane** performs the necessary interpolation for a data slice of any orientation in a 3-dimensional field.

   See MapToPlane in *IBM Visualization Data Explorer User's Reference*.

To visualize an orthogonal slice without interpolation, use **Slab**:

1. Open and execute visual program `.../Slab.net`. The image is a translucent isosurface with a colored slice (or slab) cutting through it. To visualize a slice through another part of the isosurface:
2. Open the configuration dialog box for **Slab** and change the **position** parameter value to "10."
3. Click on **OK** or **Apply**.
4. Reexecute the visual program. The position of the new slice is changed.

   See Slab in *IBM Visualization Data Explorer User's Reference*.

To create an animation that generates different slices of the data:

1. Select **Special** in the categories palette and then **Sequencer** in the tools palette.
2. Position the cursor to the right of **Slab** in the VPE.
3. Open the **Slab** configuration dialog box.
4. Click on the **position** toggle (unsetting the parameter value). The parameter field now reads "(all)" and the *third* input tab on the **Slab** icon (counting from the left) projects outward from the icon (instead of into it).
5. Click on **OK**.
6. Connect the output tab of **Sequencer** to this third input tab (i.e., "position") of **Slab**.
7. Double click on the **Sequencer** icon to display the **Sequence Control** panel.
8. Click on the frame button (**...**) to display the **Frame Control** panel.
9. Reset the limits:
   a. Click on the **min** field, type "0," and press Enter.
   b. Click on the **max** field, type "20," and press Enter.
10. Click on the Forward (►) button to play the sequence.

    See "Using the Sequencer" on page 14 in this Guide and "Sequencer" on page 297 in *IBM Visualization Data Explorer User's Reference*.

Streamlines

The **Streamline** module traces the path of a massless particle through a static velocity field.

1. Open and execute visual program `.../Streamlines3D.net`. The image is a translucent isosurface with a single streamline starting from the point [25000 5000 25000] (as specified in the **Streamline** module's configuration dialog box). This streamline can be transformed into a ribbon:
2. Select **Annotation** in the categories palette and then **Ribbon** in the tools palette.
3. Position the **Ribbon** icon below **Streamline** in the VPE canvas.
4. Disconnect **Streamline** output from **Collect** input and reconnect it to **Ribbon** input.
5. Connect **Ribbon** output to **Collect** input.
6. Reexecute the visual program. The streamline changes to a ribbon.

If you want the twist of the ribbon to represent the vorticity of the wind field:

1. Open the **Streamline** configuration dialog box.
2. Change the **flag** parameter value from "(input dependent)" to "1" and click on **OK**. **Streamline** computes the degree of twist from the vorticity of the wind field.
3. Reexecute the visual program. The twist is greater in regions of higher wind vorticity.

To make the color of the ribbon correspond to wind velocity:

1. Select **Transformation** and then **AutoColor** in the palettes.
2. Position the **AutoColor** icon between **Ribbon** and **Collect** in the VPE canvas.
3. Disconnect the **Ribbon** output from the **Collect** input and reconnect it to the first (leftmost) input of **AutoColor**.

   Note: Both input tabs can accept a connection, but the semi-highlighting indicates *required* input (i.e., the module cannot function without it).
4. Connect the first (leftmost) output of **AutoColor** to the available input of **Collect**.
5. Reexecute the visual program. Note the variation of color in the ribbon.

   See Ribbon and Streamline in *IBM Visualization Data Explorer User's Reference*.

3-D Scalar Glyphs

Scalar glyphs can represent 3-dimensional as well as 2-dimensional data.

1. Open and execute visual program `.../AutoGlyph3DScalar.net`. The spherical glyphs on the isosurface represent a subset of the data elements.
2. To visualize the entire data set:
   a. Disconnect the **Map** output from **AutoGlyph**.
   b. Connect the output of the left-hand **Import** module to the first ("data") input tab of **AutoGlyph**.
   c. Reexecute the visual program. The number of glyphs is greatly increased.

You can also create your own glyphs (both scalar and vector). For example:

- Connect the output of **Shade** to the second ("type") input tab of **AutoGlyph**.
- Reexecute the visual program. The combination of **Construct**, **Connect**, and **Shade** produces a small pyramidal glyph.

  See AutoGlyph, Connect, Construct, and Shade in *IBM Visualization Data Explorer User's Reference*.

3-D Vector Glyphs

Vector glyphs can represent 3-dimensional as well as 2-dimensional data.

1. Open and execute visual program `.../AutoGlyph3DVector.net`. The image is a set of 3-dimensional arrow glyphs on an isosurface.
2. To visualize the entire data set:
   a. Disconnect **Map** output from **AutoGlyph**.
   b. Connect the output of the left-hand **Import** module to the first ("data") input tab of **AutoGlyph**.
   c. Reexecute the visual program. The number of glyphs is greatly increased.

Volume Rendering

Volume rendering is a technique for using color and opacity to visualize the data in a 3-dimensional data set. (In contrast, surface techniques use tools like **Isosurface** and **MapToPlane** to display a 2-dimensional surface, although in 3-dimensional space.) The following are some simple examples.

1. Open and execute visual program `.../VolumeRendering.net`. As the network in the canvas shows, the color of the volume is determined by **AutoColor**. The data set contains relatively few high values (red) and relatively many low values (blue). No structure is apparent in the image.
2. Select **Transformation** and then **Equalize** in the palettes.

3. Position the **Equalize** icon between **Import** and **AutoColor**in the VPE canvas.
4. Disconnect **Import** output from **AutoColor** input and reconnect it to the first input tab ("data") of **Equalize**.
5. Connect **Equalize** output to the first input tab ("data") of **AutoColor**.
6. Reexecute the visual program. **Equalize** redistributes the data values more or less uniformly between the minimum and maximum of the data. Although the resulting image is somewhat diffuse, the structure of the data (the electron density of an imide molecule) is now visible.

AutoColor parameters can be used to add definition to the structure.

1. Delete the **Equalize** module: Click on the icon and select **Delete** in the **Edit** pull-down menu. The connections to **Import** and **AutoColor** are automatically deleted along with the icon.
2. Reconnect the **Import** output to the first input tab ("data") of **AutoColor**.
3. Open the **AutoColor** configuration dialog box.
4. Set the value of the **min** parameter to ".1" and click on **OK**.
5. Reexecute the visual program. All data values smaller than 0.1 are rendered invisible (black). The image is much darker, but still visible.
6. To increase the visibility of the data, increase the value of the **intensity** parameter in the **AutoColor** configuration dialog box to "5."
7. Click on **OK** and reexecute the visual program. The structure of image is now fairly distinct.

A color map gives you much greater control over the appearance of the image:

1. Disconnect **AutoColor** from **Image** and connect the **Color** output to the **Image** input.
2. Reexecute the visual program.
3. Bring up the Colormap Editor by double clicking on the **Colormap** icon. The color-bar, Hue, and Opacity settings are clearly reflected in the image: regions of low data values (green) and smaller regions of higher data values (red). All other data values have been rendered invisible.

   Note: To make a region or volume invisible, it is necessary to set *both* its intrinsic opacity and its color value to zero. The reason is that the volume rendering model assumes that regions emit light as well as absorb it. So even if its opacity is zero (no absorption), a region will still emit light unless its color is black ([0 0 0]).

It is interesting to contrast the volume rendering of previous images with a surface technique. For example:

1. Disconnect **Color** from **Image** and connect the **Isosurface** output to the **Image** input.
2. Reexecute the visual program. The resulting image is an isosurface representation of the structure of an imide molecule.

You can also combine surface techniques with volume rendering. For example:

1. Select **Structuring** and then **Collect** in the palettes.
2. Position the **Collect** icon above **Image** in the VPE canvas.
3. Disconnect **Isosurface** from **Image**.
4. Connect the first output tab ("mapped") of **AutoColor** to either of the **Collect** input tabs.
5. Connect the **Isosurface** output to the other **Collect** input tab.
6. Connect the **Collect** output to the **Image** input.

7. Reexecute the visual program. The result is a combination of the volume-rendering and isosurface images of the imide molecule.
8. To make the isosurfaces translucent, insert a new **Color** module (from the **Transformation** category) into the network between **Isosurface** and **Collect**. (Use the first, or "input," tab of the **Color** icon.)
9. Open the **Color** configuration dialog box and set the **opacity** parameter to ".3." (You can try other values as well.)
10. Click on **OK** and reexecute the visual program. The isosurfaces are now translucent.

3.8 Tasks and Tools

The following examples illustrate a number of techniques and tools that are helpful in using Data Explorer.

Adding Captions

The **Caption** tool allows you to control the placement, font, size, wording, and other aspects of a caption in the Image window.

1. Open visual program `.../Caption.net`.
2. Select **Open All Control Panels** in the **Windows** pull-down menu. A control panel appears.
3. Select **Execute on Change** in the **Execute** pull-down menu. When the image appears, note that the caption at the bottom of the Image window is the same as the name of the realization technique shown in the control panel.
4. Click on the option button in the control panel and select **Streamlines**. Both the image and the caption change accordingly.

Use the **Caption** configuration dialog box to change the placement and font size of the caption.

1. Double click on the **Caption** icon to open the configuration dialog box and click on the **position** toggle button.
2. Clear the **position** parameter field and type "0 1".
3. Click on **OK**. The caption moves from its default position to the top left corner of the Image window.

   Note: [0 0] specifies the bottom left corner and [1 1] the top right.
4. Reopen the Caption configuration dialog box and click on **Expand** to show the hidden parameters.
5. Click on the **height** toggle button.
6. Double click on the associated parameter field to highlight the value there.
7. Type a larger value in its place (e.g., if the value was "15," type in "20").
8. Click on **OK**. The caption type changes size. (See "Displaying and hiding parameters" on page 38.)

To change the wording of a caption, you must use the **Format** tool:

1. In the VPE window, disconnect the **Caption** input from the **Selector** interactor.
2. Select **Annotation** and **Format** in the palettes.
3. Position the **Format** icon between **Selector** and **Caption** and open the configuration dialog box.
4. Drag the cursor over "(none)" in the **template** parameter field and then type "Visualization Method: %s" in its place (%s indicates that a string will be inserted).

5. Click on **OK**.
6. Connect the second output tab of **Selector** to the second (middle) input tab of **Format**.
7. Connect the **Format** output to **Caption** and reexecute the visual program. The caption reads “Visualization Method: Streamlines.”

   See Caption and Format in *IBM Visualization Data Explorer User’s Reference*.

Adding Input Tabs to Tool Icons

Tools such as **Compute**, **Options**, and **Switch**, among others, can have a variable number of inputs. If you need more tabs than the number shown by a default icon, you can increase the number:

1. Select the tool icon to which you want to add tabs.
2. Select **Input/Output Tabs** in the **Edit** pull-down menu and select **Add Input Tab** in the cascade menu. An input tab is added to the icon.
3. Repeat Step 2 as many times as necessary.

Note: You can also remove tabs from a tool icon by following the same procedure, but select **Remove Input Tab** in Step 2.

Connecting Scattered Data Points

Many Data Explorer modules cannot be used with scattered data points that have no connections (i.e., interpolation elements). However, Data Explorer does provide two methods for creating connections between scattered data values.

1. The first method uses the **Connect** module.
   - Open and execute visual program `.../Connect.net`.

     The scattered data values in the image are created by the **Construct** module. The **Connect** module uses triangles to connect them.
2. The second method uses the **Regrid** module.
   - Open and execute visual program `.../Regrid.net`.

     The data values here are the same as those in the **Connect** example. They are mapped onto a regular grid by the **Regrid** module. The rightmost **Construct** module generates the grid.

   See Connect and Regrid in *IBM Visualization Data Explorer User’s Reference*.

Controlling Execution with Switch

Switch allows you to decide which portions of a visual program are executed (e.g., whether a data set is visualized with **Isosurface** or **MapToPlane**).

1. Open visual program `.../Switch.net`.
2. Select **Execute on Change** in the **Execute** pull-down menu. The image that appears is a streamline representation of the data set.
3. Select **Open All Control Panels** in the **Windows** pull-down menu. You can now use the selector interactor to switch from one visualization to the other.

You can modify the selector interactor and increase the number of choices.

1. Double click on the interactor to open the **Set Selector Attributes** dialog box.
2. Double click on the **Value** parameter field and type “3” in that space.
3. Double click on the **Label** parameter field and type “MapToPlane.” in that space.

Note: The integer passed from **`Selector`** to the first input tab of **`Switch`** determines what input, if any, is passed on to another module. If the integer is “0” or greater than the number of objects being passed to the module, the output is NULL. Thus, “1” selects the first input (second input tab), “2” the second input (third input tab), and so on.

4. Click on **`Add`**. A new third line should appear under the first two.
5. Click on **`OK`** to close the dialog box. If you click on the option button in the selector, you will see that it now offers a third choice. Leave the control panel open.

The third choice shown in the Selector, however, is not yet operative (select **`MapToPlane`** in the control panel and reexecute the visual program). To implement this choice, you must incorporate a third visualization in the program, such as that represented by the program segment on the right side of the VPE canvas. This segment computes a MapToPlane of temperature data.

1. Click on the **`Switch`** icon to highlight it.
2. Select **`Add Input Tab`** in the **`Edit`** pull-down menu. A new input tab is added to **`Switch`**.
3. Connect the first (“mapped”) output tab of the **`AutoColor`** icon (below the MapToPlane icon) to the new input tab of **`Switch`**.
4. Select **`MapToPlane`** in the control panel and reexecute the visual program. The **`MapToPlane`** visualization appears in the Image window.

Note: Switch selects among inputs. The corresponding module that selects among outputs is Route. Both are described in *IBM Visualization Data Explorer User’s Reference*.

Controlling Inputs: Configuration Dialog Boxes

A configuration dialog box allows you to change the parameter values of a module. To open the dialog box, double click on the module’s icon or single-click on the icon and select **`Configuration`** from the **`Edit`** pull-down menu. You can close the dialog box by clicking on **`OK`**.

Changing parameter values

A configuration dialog box displays the input parameters of a module.

You can change a parameter value by typing in a new value in the corresponding parameter field on the right side of the dialog box.

Note: If a tab is already connected to an arc, you must first disconnect the arc before typing in a new value.

Displaying and hiding parameters

Most configuration dialog boxes can be “expanded” to display “hidden” parameters for less commonly used functions. If a dialog box has hidden parameters, the **`Expand`** and **`Collapse`** buttons at the bottom of the box are enabled (i.e., their labels appear in solid type; otherwise, both labels are gray).

To display hidden parameters, click on the **`Expand`** button. To restore the dialog box to its previous state, click on **`Collapse`**.

Notes:

1. Whether a parameter is hidden or visible is determined by the associated toggle button in the **Hide** column of the dialog box.
2. The number of input tabs on an icon varies with the number of *visible* parameters in the dialog box.

Controlling Inputs: Interactors

Using a configuration dialog box to specify tool inputs can be awkward, especially if the inputs are changed frequently or if the number of inputs is large. A simpler means of controlling input values makes use of *interactors*, which appear only in Control Panels. They are represented on the VPE canvas by *stand-ins*, or icons, selected from the category and tool palettes just as tools are. The output of an interactor, like that of any tool, can be connected to one or more inputs.

For this part of Tutorial II, you will use a *scalar interactor stand-in* to control an isosurface value. A scalar interactor can control any parameter that accepts a scalar value as input. Other types of interactor (e.g., vector, integer, string) can control parameters that take the corresponding type of input.

Begin by opening visual program `.../Isosurface3D.net`.

Selecting interactors and placing stand-ins

The procedure here is essentially the same as that for selecting tools and placing icons (see “Selecting tools and placing icons” on page 22).

1. Select **Interactor** in the categories palette.
2. Select **Scalar** in the tools palette and position the cursor (now an inverted “L”) above **Isosurface**.
3. Click again. The stand-in for the **Scalar** interactor appears.

Connecting the interactor

1. Click and hold on the **Scalar** output tab and drag the cursor to the middle **Isosurface** input tab (which lights up when the cursor touches it).
2. Release the mouse button to establish a connection (represented by a rectilinear black line) between the two.

Creating a control panel

Double click on the **Scalar** icon. A control panel appears containing a scalar interactor labeled **Isosurface value**. Stepper arrowheads can be used to change this value.

Setting the interactor attributes

To set interactor attributes, you must open a **Set Attributes...** dialog box by:

- double clicking on the interactor in the control panel *or*
- selecting **Set Attributes...** in the **Edit** pull-down menu of the control panel.

When the dialog box appears, you are ready to (re)set the attribute values:

1. Click on the **Maximum** field. The value disappears.
2. Type `1` and press **Enter** to set the new value.
3. Repeat Steps 1 and 2 for **Minimum**, and change the value to `.1`.
4. Repeat Steps 1 and 2 for **Global Increment** and change the value to `.01`.

5. Repeat Steps 1 and 2 for **Decimal Places** and change the value to 2. (You can also reset this value with the stepper buttons.)
6. Click on **OK**. The dialog box closes.

When the minimum and maximum values are set, Data Explorer will prevent values outside that range from being entered.

Note: As discussed in "Data-driven Tools" on page 43, *data-driven* interactors derive their own minimum and maximum from the data itself.

Executing the program on change

1. Select **Execute On Change** in the **Execute** pull-down menu of the VPE menu bar.
2. Use the right-hand stepper arrowhead in the interactor to increase the isosurface value. As the value changes, so does the image in the Image window.

   Notes:
   a. You can also change the isosurface value by clicking on it, typing in a new value, and Pressing Enter.
   b. You can accelerate the value change by holding down the mouse button after selecting a stepper arrowhead.
   c. If you change values faster than Data Explorer can generate images, it will complete processing the current value and then "jump" to the one most recently specified, passing over any intermediate values.
3. Click on the left-hand stepper arrow to decrease the value. Again, new images appear in the Image window.
4. Select **End Execution** in the **Execute** pull-down menu.

Changing the interactor style

In this example, you will change the interactor *style* from "stepper" (as in the preceding example) to "slider."

1. Click on the scalar interactor **Isosurface value** if it is not already highlighted.
2. Click on **Set Style** in the **Edit** pull-down menu of the control panel. Another pull-down menu appears.
3. Click on **Slider**. The interactor changes appearance.
4. Select **Execute On Change** as in the preceding example.
5. Using the left mouse button, drag the slider tab to the right or left to increase or decrease the isosurface value.
6. Release the mouse button to generate an image corresponding to the new value.
7. Select **End Execution** in the **Execute** pull-down menu.

Changing the interactor label

The default label of an interactor connected to a tool is the name of the tool followed by the name of the input parameter: in this case "Isosurface value."

1. Click on the **Isosurface value** interactor if it is not already highlighted with a white border.
2. Click on **Set Label...** in the **Edit** pull-down menu. The **Set Interactor Label...** dialog box appears.
3. Double click on the **Interactor Label** field or drag the mouse cursor over the text.

4. Type in a new name and click on **OK**. The dialog box closes and the new interactor label appears in place of the previous one.

   Note: You may break the label into two or more lines by typing `\n` where you want the desired line break(s) to appear.

Creating Animations

This example demonstrates a few ways of using a sequencer.

1. Open (but do not execute) visual program `.../Animate.net`.
2. Select **Sequencer** from the **Execute** pull-down menu. Note that both the Loop and Palindrome buttons in the control panel are already activated (recessed).
3. Click on the Forward button (►): the visual program executes. The Image window opens and the image begins to run continuously through an “animation” sequence in which a data slice changes position.
4. Click on the Stop button (∎).
5. Disconnect the leftmost **AutoColor** icon from **Image** and connect the left output tab of the other **AutoColor** icon to **Image**.
6. Click on the Forward button (►) again. The animation sequence shows a set of streamlines growing longer and then shorter and then repeating the process.
7. Disconnect **AutoColor** from **Image** and connect **Isosurface** to **Image**.
8. Click on the Forward button (►). The sequence is that of a continuously changing isosurface.
9. Click on the Stop button (∎) to halt the sequence.

See Sequencer in *IBM Visualization Data Explorer User’s Reference*.

Note: In this example, a transmitter and several receivers are used to make “invisible” connections between tools. The *frame* tool below the **Sequencer** icon is a transmitter. The other *frame* tools are receivers. Receivers and transmitters belong to the **Special** category of tools (see *IBM Visualization Data Explorer User’s Reference*).

Changing the limits of the sequencer

1. If the sequence is still running, click on the Stop button (∎) to halt it.
2. Click on the ellipsis button (**...**) in the **Sequence Control** panel to open the **Frame Control** dialog box.
3. Click on the **Max** parameter field, type in “30,” and then press Enter.
4. Click on the Forward button (►) to start the sequence. The isosurface sequence now proceeds to larger isovalues (smaller isosurfaces).
5. Close both control panels by double clicking on the top left button of the **Sequence Control** box.

For Future Reference

Min and **Max** control the minimum and maximum number of frames that a sequencer can generate. **Start** and **End** are set by default to the same values, but they can take any values in the range. The **Increment** parameter controls the difference between output values (frames).

Creating and Using Macros

Macros are collections of tools that can be represented by a single icon in the VPE canvas. Macros thus allow you not only to simplify the appearance of your visual program but also to share commonly used functions between programs. This section briefly introduces the basic concepts of creating and using macros. (The topic is treated in detail in *IBM Visualization Data Explorer User's Guide*.) The general procedure for creating a macro follows on the next page.

1. Decide how many inputs and outputs your macro will have.
2. For each input, select **Special** in the categories palette and then **Input** in the tools palette.
3. Position the mouse cursor in the VPE canvas and click once to generate an **Input** icon.
4. Repeat Steps 2 and 3 for **Output**.
5. Open the configuration dialog box for each tool to give it an appropriate name, description, and default value.
6. Select any additional tools you want to include in the macro and place their icons on the canvas.
7. Connect the **Input** and **Output** icons to the appropriate tools.
8. Select **Macro Name** in the **Edit** pull-down menu and name the macro.
9. Save the macro.

For Future Reference

- To use the new macro, you must first load it into Data Explorer: select **Load Macro** from the **File** pull-down menu.

  Once loaded, the macro will be available from the tools palettes. Simply select it and place its icon on the canvas.
- For descriptions of the inputs and outputs specified when the macro was created, open its configuration dialog box.
- To see the component contents of the macro, click on the icon to highlight it and then select **Open Selected Macro** in the **Windows** pull-down menu.

The following example illustrates the use of a macro in a visual program.

1. Select **Load Macro** in the **File** pull-down menu.
2. Type `/usr/lpp/dx/samples/tutorial/SampleMacro.net` in the **Filter** field at the top of the dialog box and press Enter. The name of the macro appears under **Files** on the right side of the dialog box.
3. Click on the macro name to highlight it, and then click on **Load Macro** at the bottom of the dialog box. A new category appears in the categories palette: **Macros**.
4. Select the new category. The name of the new macro appears in the tools palette. Now you can open a visual program that uses this macro.
5. Open and execute visual program `.../UseSampleMacro.net`.

   Although the visual program looks simple, the image it produces is quite elaborate.
6. Click on the **SampleMacro** icon to highlight it and then select **Open Selected Macro** in the **Windows** pull-down menu. A new window appears, displaying the "network" of **SampleMacro**: This macro performs various operations on the output from three **Input** modules (top) and feeds the result to a single **Output** module (bottom). It is this output that is fed to the **Display** module in the visual program.

Data-driven Tools

Many of the tools in Data Explorer can be "data driven". That is, their attributes (e.g., limits) can be determined dynamically at run time from the data set being used.

Note: The attributes of data-driven tools become effective only *after* the first execution with the new data set.

Data-Driven Colormap Editor

1. Open and execute visual program `.../DataDrivenColormap.net`. The image is a color-mapped slice of data, with a color bar at the top of the Image window.
2. Select **Open All Control Panels** in the **Windows** pull-down menu and change the **Selector** value to **wind**.
3. Reexecute the visual program. The color map changes to reflect the new data set.

See Colormap in *IBM Visualization Data Explorer User's Reference*.

Data-Driven scalar interactor

Scalar, integer, and vector interactors can all be data driven. The example here is that of a scalar interactor.

1. Open and execute visual program `.../DataDrivenScalar.net`.
2. Select **Open All Control Panels** in the Windows pull-down menu and change **Format value** to **cloudwater**.
3. Leave the control panel open and reexecute the visual program. The **Isosurface value** (along with the image) is updated to reflect the new data set.

See Scalar in *IBM Visualization Data Explorer User's Reference*.

Data-driven selector

In this example, the input from **Import** to the **Selector** interactor is a data group consisting of two fields.

1. Open and execute visual program `.../DataDrivenSelector.net`. The image is a temperature field.
2. Select **Open All Control Panels** in the **Windows** pull-down menu.
3. Select **wind_velocity** in the control panel and reexecute the visual program.

The control-panel options **temperature** and **wind_velocity** are derived from the field names of the imported data.

In this network, **Inquire** determines whether or not the data set is "vector" and, if so, **Include** excludes invalid data values. If the data is not vector, the path followed goes directly from **Select** to **Switch**.

See Selector in *IBM Visualization Data Explorer User's Reference*.

Data-driven sequencer

In this example, the limits for the **Sequencer** are set automatically according to the number of elements in the x-dimension of the data set (as determined by the **Inquire** module).

1. Open (but do not execute) visual program `.../DataDrivenSequencer.net`.
2. Select **Sequencer** from the **Execute** pull-down menu. Note that both the Loop and Palindrome buttons in the control panel are already activated (recessed).
3. Click on the Forward button (▶): the visual program executes. The Image window opens and the image begins to run continuously through an “animation” sequence of two dozen frames.
4. Stop the animation sequence by:
    - Clicking on the Stop (■) or Pause (❙❙) button, *or*
    - Clicking on the Loop and Palindrome buttons to deactivate them.

See Sequencer in *IBM Visualization Data Explorer User’s Reference.*

Modules: Using AutoColor

The AutoColor module automatically colors data for you. By default, it colors data from blue (for minimum values) to red (for maximum values).

1. Open and execute visual program `.../AutoColor.net`. The image is a translucent isosurface and a colored plane.
2. Select **Sequencer** in the **Execute** pull-down menu and click on the Forward (▶) button in the **Sequence Control** panel. The colors represent wind data.
3. Click on the Stop button (■) before proceeding.

Because **AutoColor** colors each plane individually, the full blue-red range is used each time to represent the wind variation in a single plane. As a result, the same color in different planes can represent different data values and different colors can represent the same value. In order to make the color representation consistent from plane to plane:

1. Connect the output tab of the rightmost **Import** icon to the second (projecting) input tab of **AutoColor** (“min”).
2. Click on the Forward button (▶) of the Sequencer again. **AutoColor** now applies the blue-red range to the entire data set, so that the same color values consistently represent the same data values in every plane.
3. Open the **AutoColor** configuration dialog box and click on **Expand** to show the hidden parameters. (See “Displaying and hiding parameters” on page 38.)
4. Click on the **opacity** toggle button and change the value in the parameter field to “.3.”
5. Click on the Forward button (▶) of the Sequencer again and note the change in the plane.

See AutoColor in *IBM Visualization Data Explorer User’s Reference.*

Modules: Using Compute

Compute is a general purpose module for performing algebraic, trigonometric, and logical operations on data. It can also extract components from vectors, create vectors from scalar components, and cast between different data types.

1. Open and execute visual program `.../Compute.net`. The image represents wind velocity over the surface of the earth. Color values are based on the

magnitude of the wind velocity, and the small black squares and rectangles represent missing data.

2. Disconnect **Include** from **AutoColor** and connect the output of **Compute** to **AutoColor** instead.
3. Execute the visual program. Now the colors are based on the absolute value of the x-component of the wind velocity.
4. Double click on the **Compute** icon to open the configuration dialog box. In this example the module has a single input (**wind**), and the operation to be performed is to determine the absolute value of its x-component.
5. Change "abs(wind.x)" in the **Expression** field to "abs(wind.y)." and reexecute the visual program. The image changes accordingly.

Note: **Compute** is not limited to a single input, and input tabs can be added with the **Add Input Tab** function in the **Edit** pull-down menu (see "Adding Input Tabs to Tool Icons" on page 37). By default the inputs to **Compute** are labeled a, b, c,..., but you can rename them.

It is also possible to use **Compute** on components other than **data**. For example, suppose you wanted to display a regular 2-D grid of latitude and longitude on a spherical surface. You could use **Compute** to convert the x,y positions of latitude and longitude to x, y, and z positions of a spherical surface. For an example, open and execute visual program `.../WarpedGrid.net`, which generates two entirely different objects (in separate Image windows) from the same data.

See Compute in *IBM Visualization Data Explorer User's Reference*.

Modules: Using Map

The **Map** module maps one field of data onto another. The field to be mapped *to* is the first parameter; the field to be mapped *from*, the second parameter. In the most common use of this module, no other parameters are set: the positions of the first field are used as indices to the positions of the second; the associated data values from the second field are then mapped as new data values onto the first.

Components other than **positions** and **data** can also be mapped to one another.

1. Open and execute visual program `.../Map.net`. In this example, wind values are mapped onto an isosurface of cloud-water quantity and the result is colored. In addition, wind values are mapped onto a probe point and displayed as text glyphs.
2. Double click on the right-hand **Import** icon to open its configuration dialog box. The **name** parameter field contains a file name ending in "wind".
3. Change "wind" to "temperature" and click on **OK**.
4. Reexecute the visual program. The image is now a color map of temperatures on the isosurface. The number displayed is the data value of the probe point.
5. To move the probe point, first select **View Control** in the **Options** pull-down menu.
6. Next select **Cursors** in the **Mode** pull-down menu. The probe point appears as a small white square immediately to the left of the data value.
7. Drag the probe point to reposition it, and then release the mouse button. The data value is updated and displayed next to the newly positioned probe point.

See Map and Probe in *IBM Visualization Data Explorer User's Reference*.

Modules: Using Plot

Plot creates a 2-dimensional plot from x,y data. (Data Explorer expects the "positions" component to contain the x-values, and the **data** component the y-values. You can pass (import) a group of such fields to **Plot**, which plots them as multiple lines. You can also add data-point markers to the lines.

1. Open and execute visual program `.../Plot.net`. Two Image windows appear: one a color map of elevation data for the southeastern United States; the other, a plot of the elevations along the purple line shown in the color map. (The **Slab** module extracts the elevation data along the line, but **Construct** could have been used to create the line and **Map** to map the elevation data onto the line. Note that **Compute** is used to extract only the x-component of the positions.)
2. To animate the line:
   a. Select **Special** and **Sequencer** in the palettes.
   b. Position the **Sequencer** icon above **Slab**.
   c. Double click on the icon to open the **Sequence Control** panel and then single-click on the ellipsis button (**...**) to open the **Frame Control** dialog box.
   d. Set **Min** to "0" and **Increment** to "5."
   e. Click on the ellipsis button again to close the **Frame Control** dialog box.
   f. Double click on **Slab** to open its configuration dialog box.
   g. Click on the **position** toggle button to deactivate it (the associated parameter field will read "(all)") and then on **OK**.
   h. Connect the **Sequencer** output tab to the third input tab of **Slab** ("position").
   i. Click on the Forward button (►) in the **Sequence Control** panel.
3. The limits of the plot change somewhat from frame to frame because they are based on the line given for each frame. To make the limits constant:
   a. Double click on the **Plot** icon to open its configuration dialog box.
   b. Set the **corners** toggle button on and change the values in the parameter field to {[260, -6000][290, 2000]}.
   c. Click on **Expand** (to show the hidden parameters) and then on the **font** toggle button (to change the font to "roman_tser").
   d. Click on **OK** at the bottom of the dialog box.
   e. Click on the Forward button in the **Sequence Control** panel.
4. To place data-point markers on the data plot:
   a. Disconnect **Plot** from **Unmark**.
   b. Connect the **Options** output tab to the first ("input") tab of **Plot**.
   c. Reexecute the visual program. The markers appear as small open circles.
   d. To see what options were specified, double click on the **Options** icon to open its configuration dialog box. In this example, the "mark" attribute is set to "circle" and the "mark every" attribute is set to "10." Thus every tenth data point is marked with a circle.

See Plot in *IBM Visualization Data Explorer User's Reference*.

Processing Images

A variety of image processing functions are available with the **Compute**, **Filter**, and **Overlay** modules. A few of these functions are demonstrated in this example.

1. Open visual program `.../ImageProcessing.net`.
2. Select **Open All Control Panels** in the **Windows** pull-down menu. The control panel displays two selector interactors:
   - The first specifies which of three filters is to be applied to the data.

   - The second specifies which of four image-processing functions is to be performed. (Note that only the first two of these functions is affected by the options selected in the first interactor.)
3. Select **Execute on Change** in the **Execute** pull-down menu. You can now select these different functions and observe their effects.

Saving and Printing Images

Once you have created an image, you can save it to a file or send it to a printer. (See also *IBM Visualization Data Explorer User's Guide.*)

- If your visual program uses the **Image** tool (as the programs in this tutorial do), you can select **Save Image** and **Print Image** in the **File** pull-down menu of the Image window.

  When saving an image, you can choose among several formats and specify the final image size or the resolution. Similarly, you can choose format and size or resolution before printing an image on your local printer (with a command such as: `lpr -Pmy_postscript_printer`).

- If you are not using Image, you should use the WriteImage tool (**Import and Export** category).

  However, you will have to create the image with Render and display it with Display (both are from the **Rendering** category).

  Connect the output of Render to the first input tab ("input") of WriteImage. Specify the format and size, using the parameters described under WriteImage in *IBM Visualization Data Explorer User's Reference*.

3.9 Scripting Language

Sometimes it may be more convenient to use the Data Explorer scripting language instead of the Visual Program Editor to create visualizations. A common instance is that of using Data Explorer scripts in overnight batch jobs to create visualizations and save them to disk for analysis the next day.

A simple script that computes a series of isosurfaces and exports them to disk is `/usr/lpp/dx/samples/tutorial/batch_script` (this is not a visual program, so you cannot read it into the Visual Program Editor):

```
data = Import("/usr/lpp/dx/samples/data/watermolecule");
counter = 1;

macro create_iso(isovalue,counter)->(counter)
{
  isosurface = Isosurface(data,isovalue);
  filename = Format("iso%d", counter);
  counter++;
  Export(isosurface, filename);
}
```

(continued on the next page)

```
counter=create_iso(0.1, counter);
counter=create_iso(0.2, counter);
counter=create_iso(0.3, counter);
counter=create_iso(0.4, counter);
counter=create_iso(0.5, counter);
counter=create_iso(0.6, counter);

# (end of script)
```

This script first imports data from a file called `watermolecule`. It then defines a macro that takes two parameters: an isovalue and a counter. The macro returns the counter as an output. It then computes an isosurface, creates a filename (using the counter as part of the name), and exports the isosurface to that filename. The counter is also incremented. Finally the macro is called six times, with six different isovalues.

To run this script, first copy `/usr/lpp/dx/samples/tutorial/batch_script` to the directory being used. Then enter the command:

```
    dx -script < batch_script
```

or

```
    dx -script
```

and at the `dx>` prompt:

```
    include "batch_script"
```

If you want to view the exported isosurfaces, you can use visual program `/usr/lpp/dx/samples/tutorial/view_isosurfaces.net`.

Note: Visual programs are also scripts. However, if a visual program uses a macro, you must include that macro before including the visual program. For example:

```
dx> include "my_macro.net"
dx> include "program.net"
```

Chapter 4. Sample Visual Programs and Sample Macros

The directory `/usr/lpp/dx/samples/programs` contains a number of sample visual programs. In addition, a set of subdirectories categorize programs by subject (e.g., 2-D Data, Annotation, etc.). To start, you may want to look at the programs in the subdirectory /SIMPLE.

The sample visual programs listed here can be invoked, like any visual program, from the VPE. To access the available sample programs, enter: `/usr/lpp/dx/samples/programs/*.net` in the **Filter** field (of the **Open...** dialog box) and press **Enter**.

The macro descriptions are in 4.2, “Sample Macros” on page 59.

Notes:

1. For the user’s convenience, and where appropriate, some programs have been listed in more than one section.
2. Once a program has been opened, relevant descriptive and tutorial information can be accessed with:
   - the **Application Comment** option of the **Help** pull-down menu (in either the Image window or the VPE window)
   - the **Open** dialog box, by clicking on the **Comments** button.

4.1 Sample Visual Programs

Simple Visual Programs

Found in directory `$DXROOT/samples/programs/SIMPLE`

Each example program listed here illustrates a typical use of the Data Explorer module it is named for.

Arrange.net
AutoAxes.net
AutoColor.net
AutoGlyph.net
AutoGrid.net
Band.net
Caption.net
Color.net
ColorBar.net
Compute2.net
Connect.net
Construct.net
Describe.net
FaceNormals.net
GetSet.net
Gradient.net
Grid.net
Histogram.net
Include.net
Isolate.net
Isosurface.net
Light.net
Map.net
MapToPlane.net
MarkUnmark.net
Post.net
QuantizeImage.net
Reduce.net
Refine.net
Regrid.net
Route.net
Rubbersheet.net
ScaleScreen.net
ShowBoundary.net
Slab.net
Sort.net
Streamline.net
Supervise.net
Switch.net
Verify.net
VisualObject.net

2-Dimensional Data

Found in directory `$DXROOT/samples/programs/2D_DATA`

`AlternateVisualizations.net` Various ways to visualize 2-dimensional data sets.

`BandedColors.net` How to create an image with a set of constant-color bands

© Copyright IBM Corp. 1991-1997

ConnectingScatteredPoints.net How to use the Connect and Regrid modules to create connections (interpolation elements) between points.

FFT.net Computing fast and discrete Fourier transformations on sample data sets.

GeneralImport1.net, GeneralImport2.net Importing data in the general array import format.

InvalidData.net How data marked as invalid are handled by various modules.

Sealevel.net Visualizing the effect of rising sea levels on eastern U.S. coastlines.

Sort.net How to sort a field based on the y-position.

Topo.net Some ways of visualizing topographic data (in this example, two data sets sharing the same grid: elevation and a gray-scale image).

UsingDrape.net How to “drape” an image or other field onto a height map.

UsingFilter.net How to perform filtering operations on images.

UsingIsosurface.net How to create isosurfaces and contour lines.

UsingMorph.net How to perform morphological operations on images.

UsingOverlay.net Using the Overlay module to combine images.

3-Dimensional Data

Found in directory `$DXROOT/samples/programs/3D_DATA`

AnnotationGlyphs.net How to create your own glyphs for use with AutoGlyph and Glyph.

AutoColor.net A simple use of AutoColor to color a 3D field.

CappedIso.net How to “close” an isosurface at the boundary of a three-dimensional volume.

ComputeOnData.net How to perform a mathematical function on all the data values in a field.

ContoursAndCaption.net How to draw contour lines on a plane. The position of the plane is controlled by the Sequencer, and a caption shows the current position of the plane.

Distributed.net Demonstrating the module-level distributed processing capabilities of Data Explorer. The network is decomposed into three mutually disjoint subsets, each of which can be executed on a different host in a distributed network.

FlyThrough.net, **FlyThrough2.net** Two ways to create a “fly through” of data.

Imide_Potential.net Visualizing a molecule in a potential field. This program also demonstrates the use of a data-driven sequencer.

Isolate.net How to isolate connections of an object.

MappedIso.net How to map a variable onto an isosurface of another variable.

MovingCamera.net Using the Sequencer to control the position of the viewing point.

MovingSheet.net Using the Sequencer to control the position of a slice through a data set.

MRI_2.net One way to visualize a set of 2-dimensional MRI slices as a 3-dimensional volume.

PickStreamline.net Using the Pick tool to select points (on an isosurface) as the origin of streamlines in the data.

PlotLine.net, PlotLine2.net, PlotTwoLines.net Using the Plot module to display 2-dimensional plots of data values.

ProbeText.net One way to place text in an image to show data values at probe points.

RubberTube.net How to make a tube diameter vary with the data value.

SharedCamera.net How to create two images that share the same camera. One of the images is used for rotating, zooming, and resizing; the other tracks the changes.

SimplifySurface.net How to simplify a surface consisting of triangles so that fewer triangles are used.

Streamline.net Using parameters of the Streamline module to visualize a vector field.

ThunderGlyphSheet.net Visualization of simulation data from a model of a thunder storm. (A slicing surface of variable position and shape is used to intersect an isosurface of the data; the area of the isosurface is calculated and displayed.)

ThunderStreamlines.net Using the Streamline module to visualize a wind field.

Thunder_cellcentered.net Differences between position-dependent and connection-dependent (cell-centered) data.

UsingClipPlane.net Using probes to control the orientation of the clipping plane.

UsingGlyphs.net Using the AutoGlyph module to create glyphs for data values.

UsingIsosurface How to create isosurfaces and contour lines.

UsingMap.net Three ways to use the Map module. Two fields differing only in the content of their data components are used as the data-set and mapping operands of the Map module. The results are displayed in three separate windows.

UsingStreakline.net Using the Streakline module to visualize a vector-field series.

VolumeClip.net How to clip a surface in a three-dimensional volume by a plane and then close the surface by the plane.

VolumeRenderingSimple.net, VolumeRendering.net How to create a volume rendering of a 3-dimensional data set. The first example is the simplest possible visual program; the second is more complex, with a color bar to annotate the image.

WindVorticity.net Using the DivCurl module to display the vorticity of a vector field. The program also uses a shared camera and data-driven interactors.

Annotation

Found in directory `$DXROOT/samples/programs/ANNOTATION`

AnnotationGlyphs.net How to create your own glyphs for use with AutoGlyph and Glyph.

AutoAxesSpecifyTicks.net How to explicitly specify tick locations and labels for the ticks.

BandedColors.net How to create an image with a set of constant color bands.

Categorical.net How to display categorical (non-spatial) data

ContoursAndCaption.net How to draw contour lines on a plane. The position of the plane is controlled by the Sequencer, and a caption shows the current position of the plane.

Distributed.net Demonstrating the module-level distributed processing capabilities of Data Explorer. The network is decomposed into three mutually disjoint subsets, each of which can be executed on a different host in a distributed network.

ExpandedFonts.net How to use the expanded font sets of the area and roman_ext fonts.

FontPreview.net Displaying ASCII text in a Data Explorer font.

GroceryList.net Plots information from a grocery list in a number of different ways; by category, by item name, etc.

Imide_Potential.net Visualizing a molecule in a potential field. This program also demonstrates the use of a data-driven sequencer.

Legend.net Shows how to use the Legend module to associate colors with strings.

PickPlot.net Demonstrates how to use picking in a plot to extract x,y positions.

PlotGroupOfLines.net How to plot a multiple-line graph

PlotLine.net, PlotLine2.net, PlotTwoLines.net Using the Plot module to display 2-dimensional plots of data values.

PlotSpecifyTicks.net How to explicitly specify tick locations and labels.

ProbeText.net One way to place text in an image to show data values at probe points.

SalesOnStates.net Shows how to display sales or other data on a per-state basis.

`Sort.net` How to sort a field based on the y-position.

`SpecialCharacters.net` Displaying all of the characters in the “pitman” or “area” fonts supplied with Data Explorer. How to specify non-ASCII characters.

`ThunderGlyphSheet.net` Visualization of simulation data from a model of a thunder storm. (A slicing surface of variable position and shape is used to intersect an isosurface of the data; the area of the isosurface is calculated and displayed.)

`UsingGlyphs.net` Using the AutoGlyph module to create glyphs for data values.

`UsingTextAndTextGlyphs.net` How to display text data at specified locations in the space occupied by an object.

Categorical

Found in directory `$DXROOT/samples/programs/CATEGORICAL`

`Categorical.net` How to display categorical (non-spatial) data

`Duplicates.net` How to detect duplicate values in a data set using CategoryStatistics

`GroceryList.net` Plots information from a grocery list in a number of different ways; by category, by item name, etc.

`Legend.net` How to create a legend, which associates a set of colors with a set of strings

`SalesOnStates.net` How to display sales data by state

`ZipCodes.net` How to import some data associated with zipcodes using ImportSpreadsheet, then display those data on a state map. This program also illustrates the use of CategoryStatistics to compute the mean data value for each category (in this case, zipcode).

Colormap Editor

Found in directory `$DXROOT/samples/programs/COLORMAP_EDITOR`

`DataDrivenInteractors.net` Some uses of data-driven interactors.

`Imide_Potential.net` Visualizing a molecule in a potential field. This program also demonstrates the use of a data-driven Sequencer.

`StandardColormaps` Demonstrates a number of useful colormaps, depending on the type of data being viewed.

`UsingColorMaps.net` Using color maps to control the visualization of data.

`VolumeRendering.net` How to create a volume rendering of a 3-dimensional data set.

Compute

Found in directory `$DXROOT/samples/programs/COMPUTE`

(Compute is a general purpose module for performing mathematical operations on data.)

`Bounce.net` The path of a bouncing ball.

`Compute2.net` A simple visual program which demonstrates how to use the Compute2 module.

`ComputeMultiLine.net` How to pass a multiline arithmetic expression to Compute2.

`ComputeOnData.net` How to perform a mathematical function on all the data values in a field.

`WarpingPositions.net` How to use the Compute module to “warp” the positions component of a field (e.g., warping a 2-dimensional field into the shape of a cylinder or sphere).

Data-driven Interactors

Found in directory `$DXROOT/samples/programs/DATA_DRIVEN_INTERACTORS`

`DataDrivenInteractors.net` Some uses of data-driven interactors.

`DataDrivenSelector.net` How to use a data-driven selector interactor.

`Imide_Potential.net` Visualizing a molecule in a potential field. This program also demonstrates the use of a data-driven Sequencer.

Debugging

Found in directory `$DXROOT/samples/programs/DEBUGGING`

`Verify.net` How to check the internal consistency of a data object.

`VisualObject.net` Display the hierarchy of a data field.

Distributed Processing

Found in directory `$DXROOT/samples/programs/DISTRIBUTED_PROCESSING`

`Distributed.net` Demonstrating the module-level distributed processing capabilities of Data Explorer. The network is decomposed into three mutually disjoint subsets, each of which can be executed on a different host in a distributed network.

Image Processing

Found in directory `$DXROOT/samples/programs/IMAGE_PROCESSING`

`FFT.net` Computing fast and discrete Fourier transformations on sample data sets.

`MRI_1.net` One way to visualize 2-dimensional MRI slices.

`UsingEqualize.net` How to use the Equalize module to emphasize features in an image.

UsingFilter.net How to perform filtering operations on images.

UsingMorph.net How to perform morphological operations on images.

UsingOverlay.net Using the Overlay module to combine images.

Importing Data

Found in directory `$DXROOT/samples/programs/IMPORTING`

Categorical.net How to import some data associated with state abbreviations using ImportSpreadsheet, then plot the average data for each state.

Duplicates.net checks a simple table of data and checks for duplicate state names.

GeneralImport1.net, GeneralImport2.net Importing data in the general array import format.

ImportExternalFilter.net How to use the external-filter option of the Import module to import data in file formats other than Data Explorer.

ZipCodes.net How to import some data associated with zipcodes using ImportSpreadsheet, then display those data on a state map. This program also illustrates the use of CategoryStatistics to compute the mean data value for each category (in this case, zipcode).

Interface Control

Found in directory `$DXROOT/samples/programs/INTERFACE_CONTROL`

DialogStyle.net Using dialog-style control panels. This program is intended to run with Data Explorer in image mode (i.e., started with `dx -image`).

InterfaceControl1.net Using ManageControlPanel and ManageColormapEditor to open and close control panels and the Colormap editor according to the setting of the Selector interactor (alternating between two realization techniques).

InterfaceControl2.net Using ManageImageWindow to close a plot image window when a toggle interactor requires it.

InterfaceControl3.net Using ManageControlPanel to open and close different multiple panels.

Looping

Found in directory `$DXROOT/samples/programs/LOOPING`

Accumulate.net How to use Get and Set to accumulate objects (in this case, slabs selected from a 3-dimensional volume).

Bounce.net The path of a bouncing ball.

SimpleGetSetLoop.net How to create a program loop with GetLocal/SetLocal, GetGlobal/SetGlobal, and Done.

Miscellaneous

Found in directory `$DXROOT/samples/programs/MISC`

CensusData.net How to visualize census data on a map of the United States.

ExampleSMP.net How Partition should be incorporated in a visual program so that Data Explorer SMP can run it in parallel on SMP (symmetric multiprocessor) machines.

Factorial.net How to compute N factorial using looping.

HomeOwn.net Illustrates the use of the MapOnStates macro to plot home ownership in the United States over time.

ImageTool.net Using interactors to control different aspects of the Image tool (e.g., AutoAxes and background color).

Image_wo_UI.net Demonstrates the Image2Macro, which implements much of the functionality of the Image tool with SuperviseWindow, SuperviseState, and Display. Thus this macro shows how you could build your own custom direct interactors independent of the Data Explorer user interface.

IndependentlyArrange.net Illustrates how to independently arrange a number of interactive windows within a single larger window using the SuperviseWindow and SuperviseState modules.

InsetImage.net Illustrates how to inset an independently interactive window within a larger window.

Interop.net One way of making Data Explorer modules work together. Data can be mapped onto objects at different points in the visualization to achieve desired results.

InvalidData.net How data marked as invalid are handled by various modules.

Majority.net The difference between row and column majority when using the general array format.

ManipulateGroups.net How to use ChangeGroupType and ChangeGroupMember to restructure groups.

MRI_1.net One way to visualize 2-dimensional MRI slices.

MRI_2.net One way to visualize a set of 2-dimensional MRI slices as a 3-dimensional volume.

MultipleDataSets.net Using the Inquire module and data-driven interactors to make visual programs more flexible. A relatively “generic” program that can be used with a variety of different input data sets.

PickStreamline.net Using the Pick tool to select points (on an isosurface) as the origin of streamlines in the data.

ScatterData.net Some ways to visualize scattered data

`Supervise.net` Demonstrates how to use SuperviseWindow, SuperviseState, and Display together in a simple visual program.

`Topo.net` Some ways of visualizing topographic data (in this example, two data sets sharing the same grid: elevation and a gray-scale image).

`UsingAttributes.net` Using attributes in a Data Explorer format file. (In this example, attributes added to the file to indicate date and source are used to caption the image.)

`UsingDrape.net` How to “drape” an image or other field onto a height map.

`UsingEqualize.net` How to use the Equalize module to emphasize features in an image.

`UsingMessage.net` Using the Message module to present information to the user of a visual program.

`UsingMultiGrids.net` Some of the differences between multigrid groups and generic groups.

`UsingParse.net` Using the Parse module to extract information from a string.

`UsingSwitchAndRoute.net` Using the Switch and Route modules to control the execution of a visual program.

Probes

Found in directory `$DXROOT/samples/programs/PROBES`

`PlotLine2.net` Using the Plot module to display 2-dimensional plots of data values.

`ProbeText.net` One way to place text in an image to show data values at probe points.

`UsingProbes.net` Using probes to control the position of a cutting plane through a 3-dimensional data set.

`UsingStreakline.net` Using the Streakline module to visualize a vector-field series.

Rendering

Found in directory `$DXROOT/samples/programs/RENDERING`

`FatLines.net` How to antialias lines and create multiple pixel width lines in hardware rendering.

`Isolate.net` Demonstrates an alternative to volume rendering using the Isolate module.

`SharedCamera.net` How to create two images that share the same camera. One of the images is used for rotating, zooming, and resizing; the other tracks the changes.

`TextureMapOpenGL.net` Using texture mapping (available only with a hardware adapter that supports OpenGL) to map an image onto a surface.

UsingClipPlane.net Using probes to control the orientation of the clipping plane.

UsingLights.net How to use lights to illuminate the object in an image.

UsingShade.net Controlling the shading properties of an object.

VolumeRenderingSimple.net, VolumeRendering.net How to create a volume rendering of a 3-dimensional data set. The first example is the simplest possible visual program; the second is more complex, with a color bar to annotate the image.

Scattered Data

Found in directory `$DXROOT/samples/programs/SCATTERED`

AnnotationGlyphs.net How to create your own glyphs for use with AutoGlyph and Glyph.

ConnectingScatteredPoints.net How to use the Connect and Regrid modules to create connections (interpolation elements) between points.

ScatterData.net Some ways to visualize scattered data

Sequencer

Found in directory `$DXROOT/samples/programs/SEQUENCER`

ContoursAndCaption.net How to draw contour lines on a plane. The position of the plane is controlled by the Sequencer, and a caption shows the current position of the plane.

FlyThrough.net, **FlyThrough2.net** Two ways to create a “fly through” of data.

Imide_Potential.net Visualizing a molecule in a potential field. This program also demonstrates the use of a data-driven Sequencer.

MRI_1.net One way to visualize 2-dimensional MRI slices.

MovingCamera.net Using the Sequencer to control the position of the viewing point.

MovingSheet.net Using the Sequencer to control the position of a slice through a data set.

Sort.net How to sort a field based on the y-position.

SpecialCharacters.net Displaying all of the characters in the “pitman” or “area” fonts supplied with Data Explorer. How to specify non-ASCII characters.

Streamline.net Using parameters of the Streamline module to visualize a vector field.

4.2 Sample Macros

Found in directory `$DXROOT/samples/macros`.

`ArrangeMemberMacro.net` Allows you to independently arrange a number of interactive windows within a larger window.

`AutoScaleMacro.net` Automatically scales an object to a user-specified aspect ratio. It is used by the ScatterData example visual program.

`BandColorsMacro.net` "Bands" a 2-dimensional data set and applies a user-specified set of colors to the bands. It is used by the BandedColors example visual program.

`CappedIsoMacro.net` caps an isosurface by the boundary of the data. Used by CappedIso.net.

`ClipSurfaceMacro.net` clips a surface by a plane. Is used by the macro ClipVolumeMacro.net.

`ClipVolumeMacro` clips a surface in a volume by a plane, capping the hole with a plane from the volume. Can be nested. Is used by VolumeClip.net.

`ConvertColorNameListMacro.net` Converts a list of color names to a list of RGB vectors. It is used by the BandedColorsMacro macro.

`DrapeMacro.net` drapes one 2-dimensional field over another, using one of them to deform the surface.

`FactorialMacro.net` Used by Factorial.net to compute a factorial.

`FormatListMacro.net` Formats a list of values into a list of strings.

`Image2Macro.net` Implements much of the functionality of the Image tool with SuperviseWindow, SuperviseState, and Display. Used by Image_wo_UI.net.

`InsetImageMacro.net` Allows you to inset an independently interactive window within a larger window.

`InterpolatePositionsMacro.net` Used by the FlyThrough example visual program (see "3-Dimensional Data" on page 50). It interpolates data values within a list of 3-dimensional positions.

`MakeLineMacro.net` Used by the PlotLine2 example visual program (see "3-Dimensional Data" on page 50).

The macro takes as input a data field, two points defining the start and end of a line, and the number of samples to be placed along the line. The macro has two outputs:

- a line that can be displayed in the Image window
- the data to be plotted, which has two components:
  - the "positions" component is the distance along the line
  - the "data" component is the data value at each position.

`Make3DFieldMacro.net` Given three fields with scalar data components, creates an output field where the x's are the data components of the first field, the y's are the data components of the second field, and the z's are the date components of the third field. It is used by the ScatterData example visual program.

`MapOnStatesMacro.net` Allows you to map data onto a map of the United States. It is used by the CensusData and ZipCodes example visual programs.

`MatteMacro.net` Used by Bounce.net. It mattes two images together.

`PickPlotMacro.net` Extracts the transformed x,y position from a picked point in a plot.

`UnsquishGlyphMacro.net` Provides a simple solution for the “squished glyph” syndrome, when data are scaled before rendering. It is used by the ScatterData example visual program.

Chapter 5. Importing Data

5.1 General Array Importer 63
  Describing the Data 63
  Creating a Header File 66
  Some Notes on General Array Importer Format 66
5.2 Importing Data: Header File Examples 67
  Record Style: Single-Variable Data 67
  Record Style: Multivariable Data 75
  Columnar Style 81
5.3 Header File Syntax: Keyword Statements 85
  file 86
  grid 87
  points 87
  block 88
  dependency 88
  field 88
  format 89
  header 89
  interleaving 90
  layout 92
  majority 92
  recordseparator 92
  series 93
  structure 94
  type 94
  positions 94
  end 96
5.4 Data Prompter 96
  For Future Reference 97
  Supported Formats 97
  Initial Dialog Box 98
  Simplified Data Prompter 101
  Full Data Prompter 103
5.5 Data Prompter Browser 109
  Starting the Browser 109
  Browser Menu Bar 109
  Browser Text Window 111
  Browser Offset Area 111
5.6 Using the Header File to Import Data 111

© Copyright IBM Corp. 1991-1997

Importing data into Data Explorer is the first step in creating a visualization of that data. Data Explorer supports the importation of a number of data formats: General Array Importer, Data Explorer native, CDF, netCDF, and HDF (see Appendix B, “Importing Data: File Formats” on page 241 in *IBM Visualization Data Explorer User’s Guide*). The General Array Importer is discussed here not only because it can import a variety of data types but because its supporting interface makes it useful to the broadest range of users. This interface consists of the Data Prompter, for describing the data to be imported, and the Data Browser, for viewing the data.

This chapter deals with the importation of data in the following sections:

- 5.1, “General Array Importer” on page 63
- 5.2, “Importing Data: Header File Examples” on page 67
- 5.3, “Header File Syntax: Keyword Statements” on page 85
- 5.4, “Data Prompter” on page 96
- 5.5, “Data Prompter Browser” on page 109

An Important Note on Fields

Importing data into Data Explorer requires some knowledge of the Data Explorer data model and at least a working knowledge of a *field*.

Fields are the fundamental objects in the Data Explorer data model. A field represents a mapping from some *domain* to some *data space*. The domain of the mapping is specified by a set of *positions* and (generally) a set of *connections* that allow interpolation of data values for points between positions. Positions represent what can be thought of as (and often really are) locations in space; the data are the values associated with the space of the positions. The mapping at all points in a domain (not just those specified by the given positions) is represented implicitly by specifying that the data are dependent on (located at) the sample points or on the connections between points.

This simple abstraction is sufficient for representing a wide range of information. For example, you can describe 3-dimensional volumetric data whose domain is the region specified by positions and whose data space is the set of values associated with those positions. The domain of a 2-dimensional image on a monitor screen is a set of pixel locations, and the data space consists of the pixel color. For 2-dimensional surfaces imbedded in 3-dimensional space (e.g., traditional graphical models) the domain may be a set of positions on the surface, and the data space a set of data values on that surface.

In Data Explorer the positions and data are said to be *components* of a field, and every field must contain at least a “positions” component and a “data” component. Fields may also contain other components (e.g., “connections”). Thus a Data Explorer field consists of data and the additional components needed to describe that data so that Data Explorer can process it.

(cont.)

An Important Note on Fields (cont.)

Components are represented as arrays of numbers with some auxiliary information specifying *attributes* (e.g., type of data dependency). The syntax of defining fields in the General Array format is described in 5.3, “Header File Syntax: Keyword Statements” on page 85. The various components are described in *IBM Visualization Data Explorer User’s Guide*.

5.1 General Array Importer

Describing the Data

To import data through the General Array Importer, you must be able to answer the following questions.

1. What are the independent and dependent variables? For example, if temperature and wind velocity are measured on a latitude-longitude grid, then latitude and longitude are the independent variables, temperature and wind velocity the dependent variables. In the case of resistance measurements versus the voltage applied to a semiconductor, voltage is the independent variable and resistance the dependent variable.

   Components and Variables

   In Data Explorer terminology, the values of the independent variable constitute the “positions” component of a data field. In the examples above, the first independent variable consists of locations in space and the second does not, but both would be represented as “positions” in a data field. The independent variable is *always* represented by the “positions” component.

   The values of the dependent variable constitute the “data” component.

2. What is the dimensionality of the positions and data components? In the first example above, latitude and longitude are represented by 2-dimensional positions, the temperature by scalar data, and the wind velocity by 2- or 3-dimensional vectors. In the second example, voltage is represented by 1-dimensional positions and the resistance by scalar data.

3. How is the independent variable (the set of positions) to be described? By a regular grid (which can be completely described by an origin and a set of deltas) or by an explicit list (which may or may not be part of the data file)? For example, data measurements might be on a grid of 1-degree increments in latitude and 5-degree increments in longitude; voltage levels might be a set of unrelated values stored with the resistances in the data file.

4. How are the positions connected to one another, if they are connected? For example, a regular grid of positions might be connected by a regular grid of connections (lines, quads, or cubes). The connections specify how data values should be interpolated between positions. Positions that are explicitly specified (i.e., not regular) can also be connected by a regular grid of connections (e.g., if the grid is deformed, or warped). See Figure 11 on page 64.

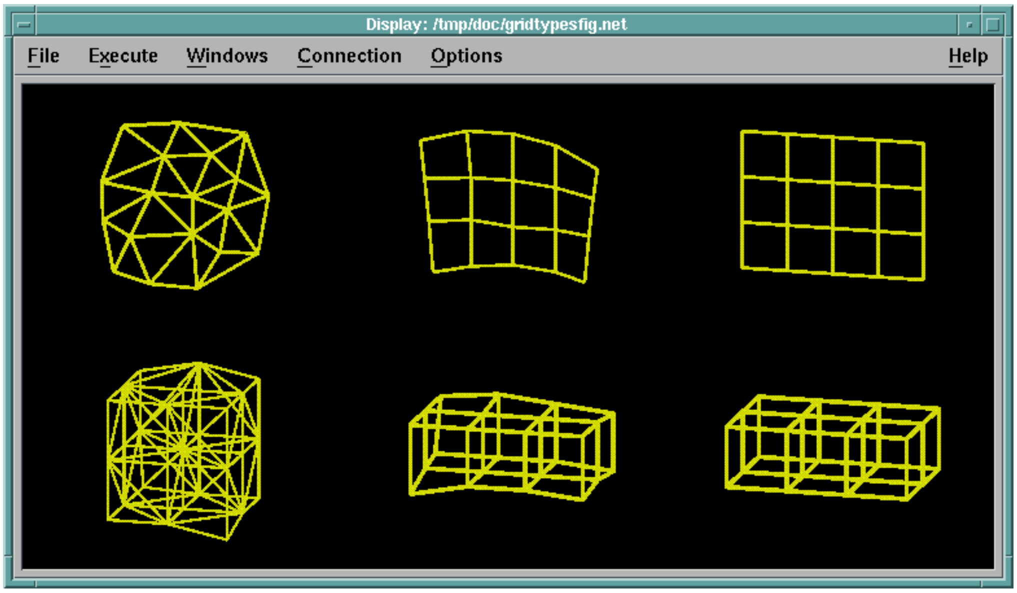

Display: /tmp/doc/gridtypesfig.net
File
Execute
Windows
Connection
Options
Help

Figure 11. Examples of Grid Types. The three grids in the top row represent surfaces; those in the bottom row, volumes. Reading from left to right, the three types of grid are: irregular (irregular positions, irregular connections), deformed regular (irregular positions, regular connections), and regular (regular positions, regular connections),

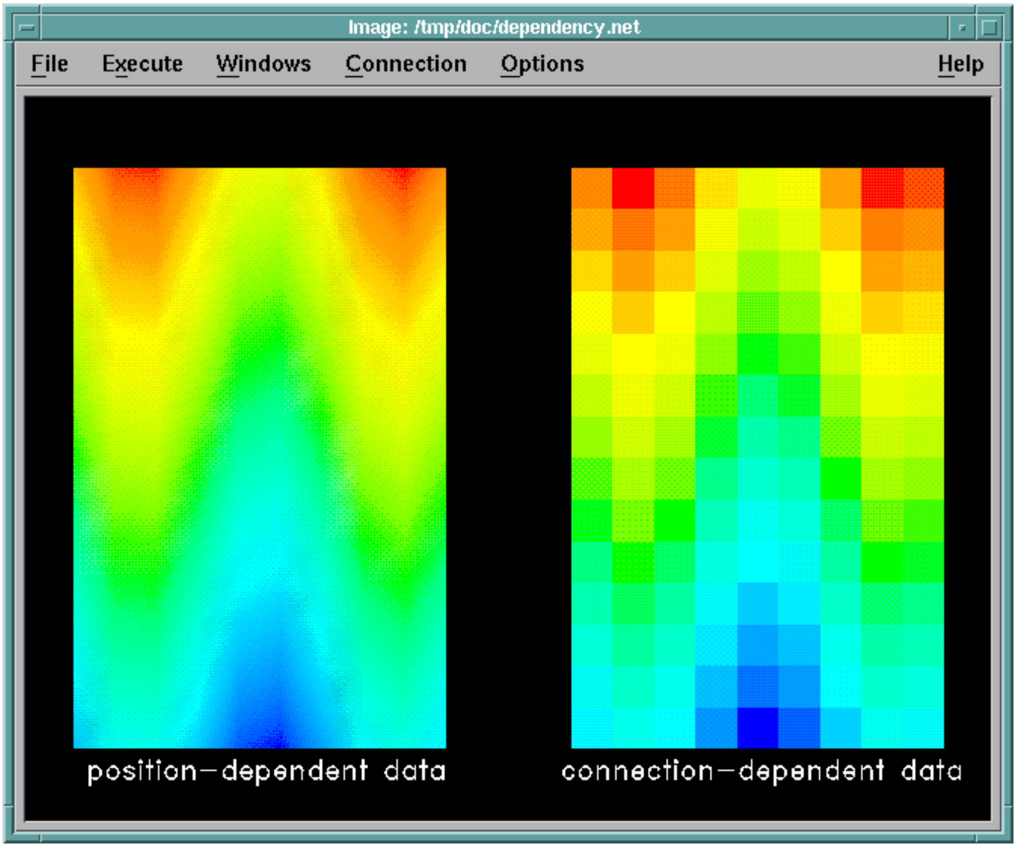


Image: /tmp/doc/dependency.net
File
Execute
Windows
Connection
Options
Help
position–dependent data
connection–dependent data

Figure 12. Examples of Data Dependency. In the visualization on the left, data correspond one-to-one with positions. Other data values (and colors) are interpolated linearly between positions. In the visualization on the right, the elements connecting positions are quads. Data (and colors) correspond one-to-one with, and are constant within, each quad.

Note: The General Array Importer supports only regular connections (lines, quads, and cubes) or scattered data. For irregular connections such as triangles or tetrahedra, you can use the Data Explorer native format to import your data. (See *IBM Visualization Data Explorer User's Guide*.)

5. What is the format of the stored data values, ASCII or binary? Are they floating point, integer, signed or unsigned byte, etc.?
6. Are the data dependent on "positions" or on "connections"? That is, are the data values associated one-to-one with positions or with the connections between positions? See Figure 12 on page 64. (Data associated with connections are often referred to as "cell-centered.") With position-dependent data, values between positions are interpolated within the connection element. With connection-dependent data, values are assumed to be constant within the connection element.
7. Do these data values represent "series data" or do they constitute only a single frame of data? In the example of resistance levels versus voltage, data may exist for each of a number of different doping levels. Each doping level could be considered a single data field and the collection treated as a series.
8. Is the data in "record" or "spreadsheet" style? (See Figure 14 on page 66.)
9. If the data are on a grid, what is the order of the data items with respect to the grid? Is it column majority (first index varies fastest) or row majority (last index varies fastest)? (See Figure 13.)
10. What kind of embedded text (comments, etc.) in the data file must be "skipped" when the data values are read?

With the answers to these questions, you can now use the General Array Importer to describe your data.

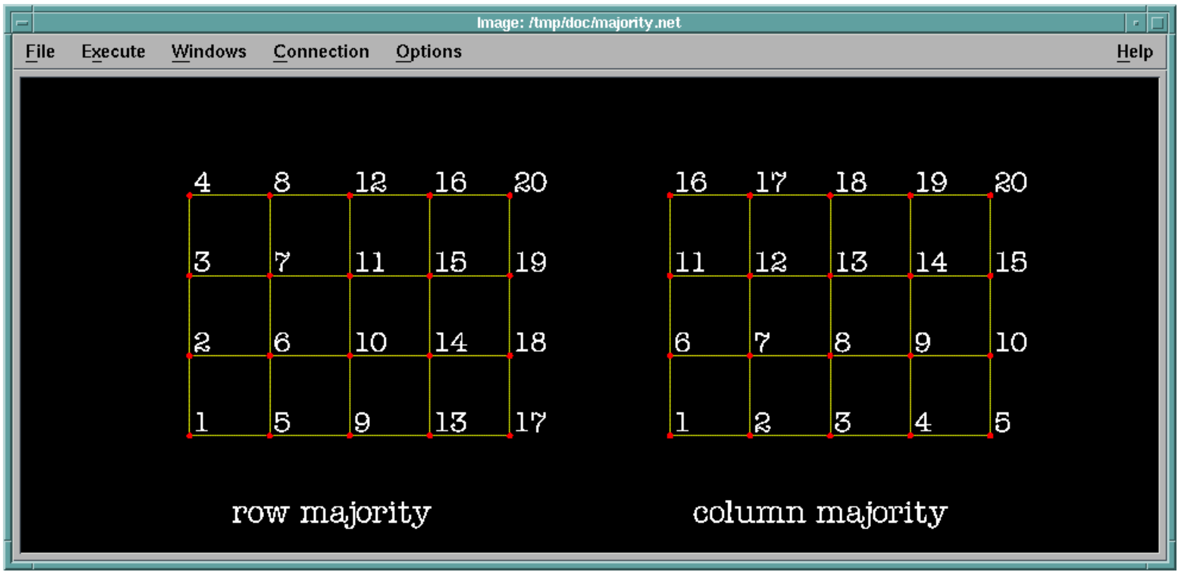


Figure 13. Row- versus Column-Majority Grids. The two grids shown here are generated from the same data file, consisting simply of the numbers 1, 2, 3, ..., 20. The associated header files differ only in the specification of the grids' majority.

Creating a Header File

The General Array Importer uses a “header file” to describe the structure and location of data to be imported. This file consists of *keyword* statements that identify important characteristics of that data (including grid structure, format, and data type, along with the path name of the file containing the data).

A header file can be created with a text editor or, more easily, with the Data Prompter, which prompts for the necessary information. (See 3.3, “Importing Data” on page 25 for an example that uses the Data Prompter and 5.4, “Data Prompter” on page 96 for a detailed description of how to use it.) The Data Prompter also checks for incorrect syntax, such as conflicting keywords (see 5.3, “Header File Syntax: Keyword Statements” on page 85).

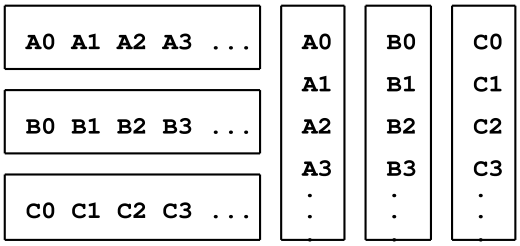
Once a header file has been created, the data it describes can be imported into Data Explorer by the Import module. To identify a header file to Data Explorer through the Import dialog box:

1. Enter the path name of the header file in the **`name`** parameter field.
2. Enter “general” in the **`format`** parameter field. (If the file has the extension “.general,” it is not necessary to specify the format to Import. Header files created with the Data Prompter are automatically given this extension.)

Some Notes on General Array Importer Format

The General Array Importer imports ASCII or binary data that is organized in one of two general “styles”: *block* or *columnar*. Block style requires that the data be organized in records, or blocks. Columnar style requires that the data be organized in vertical columns (see Figure 14).

Figure 14. Block and Columnar Styles of Data Organization. The three horizontal data blocks at left illustrate the block style; the three vertical columns at right, the columnar style. A, B, and C represent separate variables.



The following set of FORTRAN I/O statements generate a record-style data file:

```
      write(15,20) A(i),i=1,100
      write(15,20) B(i),i=1,100
      write(15,20) C(i),i=1,100
 20   format(10(f10.3))
```

An equivalent example in C is shown on the next page.

```
      for(i=0; i<100, i++) printf("%10.3f",A[i]);
      for(i=0; i<100, i++) printf("%10.3f",B[i]);
      for(i=0; i<100, i++) printf("%10.3f",C[i]);
```

The following FORTRAN I/O statement produces a columnar-style data file:

```
      write(15,10) (A(i),B(i),C(i),i=1,100)
 10   format(3(2x,f10.3))
```

An equivalent example in C is:

```
for (i=0; i<100; i++)
  printf("  %10.3f  %10.3f  %10.3f\n",A[i],B[i],C[i]);
```

For both the block and columnar styles, the information in the file can be positions as well as data. The data can be:

- scalar or vector
- a time series
- gridded or scattered (for gridded data the grid structure can be regular or warped, but the connection elements must be regular; i.e., lines, quads, or cubes)
- position dependent (associated with the grid positions) or connection dependent (associated with the grid connections).

5.2 Importing Data: Header File Examples

The examples in this section are divided into three groups: single variable (a simplified case of record style), record style, and spreadsheet style. A review of these examples will provide a good grounding in the use of the Data Prompter and the creation of header files for importing data with the General Array Importer.

The examples in the first group are generally more detailed than those in the second and third groups. Since examples often build on previous examples, it is recommended that you start at the beginning of a group.

The instructional sequence in each example begins with the initial dialog box of the Data Prompter. Most examples use the Data Prompter to create a header file and each example shows the header file produced. (For the syntax of keyword statements in a header file, see 5.3, “Header File Syntax: Keyword Statements” on page 85.) The command that invokes the Data Prompter and generates the initial dialog box is:

```
dx -prompter
```

Record Style: Single-Variable Data

It is recommended that you treat the first four examples as a unit and review them in sequence.

Example 1. Scalar Data on a Regular Grid

This example illustrates how a simple floating-point scalar field, on a regular grid, might be imported from a text file named “record_scalar”. The origin of the grid is [1 3 2], with deltas of 0.5, 1, and 0.75 in the x, y, and z directions respectively.

1. In the Data Prompter initial dialog box, choose Grid or Scattered File; then select the leftmost button in the row labeled **Grid type** (regular grid).
2. Confirm that **Number of variables** is set to “1” and that the **Single time step** toggle button is activated.
3. For **Data organization**, confirm that the **Block** (i.e., record) toggle button is activated.
4. Click on **Describe Data** to bring up the “simplified” prompter.
5. Enter the path name of the data file in the **Data file** field:

`/usr/lpp/dx/samples/data/record_scalar`

6. Browse the data file if you like by choosing **Browser** from the ellipses to the right of the **Data Field** field.
7. Enter the values "5," "8," and "6," in that order, in the first three **Grid size** fields. The number of data values is 240 (i.e., 5 × 8 × 6).
8. Confirm that **Data format** is set to ASCII (text).
9. Confirm that **Data order** is set to Row.
10. Finally, set the first three origin-delta pairs (in the **Grid positions** section of the prompter) to: 1.0, 0.5; 3.0, 1.0; and 2.0, 0.75, in that order.

    Since the data order is Row (i.e., last index varies fastest), the first six data values are associated with positions [1 3 2], [1 3 2.75], [1 3 3.5], [1 3 4.25], [1 3 5], and [1 3 5.75]. (If the data are stored so that "last index varies slowest," **Data order** should be set to Column.)
11. To save the header file:
    a. Select **Save As...** from the **File** pull-down menu.
    b. When the **Save a Data Prompter Header File...** dialog box appears, position the cursor in the **Selection** field (at the point indicated by the carat).
    c. Enter the name of the file (`record_scalar`).
    d. Click on **OK**.

    The file is saved in your current directory with the extension ".general" and should look like:

```
file = /usr/lpp/dx/samples/data/record_scalar
grid = 5 x 8 x 6
format = text
interleaving = record
majority = row
field = field0
structure = scalar
type = float
dependency = positions
positions = regular, regular, regular, 1.0, 0.5, 3.0, 1.0, 2.0, .75

end
```

Note the information that you have supplied directly (lines 1, 2, and 10). You can visualize the data file using the **Visualize Data** button in the initial Data Prompter window.

Example 2. Cell-centered Data

This example involves modifying the header file created in Example 1. The important difference is that the data here is cell-centered (connection dependent): instead of 240 data values (one for each of the 5 x 8 x 6 positions), there are 140 values (one for each of the 4 x 7 x 5 connections). The format is binary.

1. In the Data Prompter initial dialog box, click on **Grid or Scattered File**, then **Describe Data**, to bring up the simplified prompter.
2. Select **Open** from the **File** pull-down menu. A new dialog box appears called "Open a Data Prompter Header...."
3. In the **Directories** column, highlight the directory in which you saved the `record_scalar` header file (if it is not already highlighted).

4. Highlight `record_scalar.general` in the **File** column, and then click on **OK**. The simplified prompter now displays the information for the `record_scalar` header file.
5. Change the path name in the **Data file** field to:

   `/usr/lpp/dx/samples/data/record_depconnections`
6. Select **Full prompter** from the **Options** pull-down menu.
7. When the **Full prompter** dialog box appears, change **Dependency** from “positions” to “connections.” To confirm this change, click on **Modify** at the bottom of the dialog box (note the instruction there).
8. Repeat Example 1, Step 10 (see above), to save the header file, which should look like:

```
file = /usr/lpp/dx/samples/data/record_depconnections
grid = 5 x 8 x 6
format = text
interleaving = record
majority = row
field = field0
structure = scalar
type = float
dependency = connections
positions = regular, regular, regular, 1, .5, 3, 1.0, 2.0, .75

end
```

Note the information that you have supplied directly or changed (lines 1, 2, 3, and 9).

Example 3. Data with Header information

A data file may contain descriptive information in addition to the data to be imported. To import only the data, therefore, it is necessary to “skip” such information when the file is read. The **header** keyword statement enables you to do just that, by specifying a number of bytes or lines to be skipped or a string to be searched for. For example, suppose the scalar data field of Example 1 had 3 lines of descriptive text preceding the data.

1. As in Example 2 (Steps 2 through 4), open the `record_scalar` header file.
2. Change the path name in the **Data file** field to:

   `/usr/lpp/dx/samples/data/record_withheader`
3. Activate the **Header** toggle button and then click on the option button just to the right of it.
4. From the list that appears, select **# of lines** and enter the value “3” in the associated text field.
5. Repeat Example 1, Step 10 (see To save the header file on page 68), to save the header file, which should look like:

```
file = /usr/lpp/dx/samples/data/record_withheader
grid = 5 x 8 x 6
format = text
interleaving = record
majority = row
header = lines 3
field = field0
structure = scalar
type = float
dependency = positions
positions = regular, regular, regular, 1, .5, 3, 1.0, 2.0, .75

end
```

Note the addition of a **`header`** keyword statement (line 6).

Example 4. Naming a Field

By default, the Data Prompter names data fields in numerical order: field0, field1, and so on. But a data field can be named with a **`field`** keyword statement.

Once the data are imported into Data Explorer, you can, for example, extract the name (using the Attribute module) and include it in a caption (using the Caption module). So if there are two types of data (e.g., temperature and pressure), each can be automatically and appropriately labeled with an identifying name, thereby "tagging" the associated data for future reference. As a result, it is also possible to import a field *by name* when there is more than one field.

For this example suppose that the data in Example 1 are temperature values (see To save the header file on page 68).

1. Open the `record_scalar` header file, as in Example 2.
2. In the right-hand panel of the prompter, change **`Field name`** from "Field0" to "Temperature." To confirm this change, click on **`Modify`** at the bottom of the dialog box (note the instruction there).
3. Repeat Example 1, Step 10 (see To save the header file on page 68), to save the header file, which should look like:

```
file = /usr/lpp/dx/samples/data/record_scalar
grid = 5 x 8 x 6
format = text
interleaving = record
majority = row
field = Temperature
structure = scalar
type = float
dependency = positions
positions = regular, regular, regular, 1, .5, 3, 1.0, 2.0, .75

end
```

Note the change in the **`field`** keyword statement (line 6).

Example 5. Deriving Grid Information from a Data File

Being able to derive grid information directly from a data file is particularly useful if you import data with a standard format but with grid dimensions that vary from data set to data set. For example, if the first line of the data file is:

```
dimensions 100 300
```

you can use any of the following `grid` keyword statements to obtain the grid dimensions from the data file.

- `grid = lines 0, 11, 3, 1, 3`

  This statement says
  1. Skip 0 (zero) lines of the file.
  2. Skip 11 characters (the word "dimensions" and one space).
  3. Read the first dimension from 3 characters.
  4. Skip 1 character.
  5. Read the second dimension from 3 characters.
- `grid = bytes 11`

  This statement says to skip 11 characters and begin reading.
- `grid = marker "dimensions"`

  This statement says to start reading after the string marker "dimensions."

See "grid" on page 87. See also B.1, "General Array Importer: Keyword Information from Data Files" on page 242 in *IBM Visualization Data Explorer User's Guide*.

Note: This derivation feature is not available with the Data Prompter.

Examples 6 and 7. Vector Data

The General Array Importer supports three representations, or "styles," of vector data: record, record-vector, and series-vector. The first two are illustrated here. For the third, see "interleaving" on page 90.

Which representation matches the data depends on a characteristic called *interleaving*. In record interleaving, the data for each vector component are stored together in individual blocks (e.g., X_0, X_1,..., X_n, Y_0, Y_1,..., Y_n). In record-vector interleaving, the components of each vector are stored consecutively (e.g., X_0Y_0, X_1Y_1,..., X_nY_n).

The following pair of examples illustrates the differences between the two representations and between the header files used to import them. The header files are identical in that they both specify a unit 2-vector that parallels the x-axis and is defined on a 5 x 4 regular grid. That is, the data consists of 20 instances of the vector [1 0].

In Example 7, the interleaving style of the data file is *record*:

```
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0
```

1. In the Data Prompter simplified dialog box, select **New** in the **File** pull-down menu (this selection clears the dialog box of any information from a previous file).
2. Enter the path name of the data file in the **Data file** field:

   `/usr/lpp/dx/samples/data/record_vectordata1.`

3. Enter the values "5" and "4," in that order, in the first two **Grid size** fields. (Note that the origin-delta values default to [0 1].)
4. Add a field to the Field list by typing a name (e.g. field0) in the **Field name** text field, and then pressing the **Add** button.
5. Change **Type** (right-hand panel) to "int."

   Note: To implement this change, you must click on **Modify** at the bottom of the dialog box. However, you can delay implementation to Step 5, and implement both steps at the same time.
6. Change **Structure** (right-hand panel) to "2-vector." To implement this change, click on **Modify** at the bottom of the dialog box (note the instruction there).
7. Step 5 activates the **Vector interleaving** button. Select $X_0,X_1,...,X_n,Y_0,Y_1,...,Y_n$ (the notation used for record style).
8. Repeat Example 1, Step 10 (see To save the header file on page 68), to save the header file, which should look like:

```
file = /usr/lpp/dx/samples/data/record_vectordata1
grid = 5 x 4
format = text
interleaving = record
majority = row
field = field0
structure = 2-vector
type = int
dependency = positions
positions = regular, regular, 0, 1, 0, 1

end
```

In Example 7, the interleaving style of the data file is *record-vector*:

```
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
```

1. If you have not closed the simplified dialog box from Example 5, all you need do is:

   a. Change the path name for the data file (in the **Data file** field) to:

      `/usr/lpp/dx/samples/data/record_vectordata2.`

   b. For **Vector interleaving**, select $X_0Y_0,X_1Y_1,...,X_nY_n$ (the notation used for record-vector style).

   Otherwise you can repeat Example 6 (or open the header file `record_vectordata1`) and make the appropriate changes.

2. Repeat Example 1, Step 10 (see To save the header file on page 68), to save the new header file, which should look like the one shown at the top of the next page.

```
file = /usr/lpp/dx/samples/data/record_vectordata2
grid = 5 x 4
format = text
interleaving = record-vector
majority = row
field = field0
structure = 2-vector
type = int
dependency = positions
positions = regular, regular, 0, 1, 0, 1

end
```

Note: If the interleaving is not specified, the default is record-vector.

Example 8. Series Data

This example illustrates how a 7-step time series of a single scalar field might be imported. The field is on a regular 5 × 5 grid, the data are connections dependent, and the style is record.

```
# Time-Series Data
Time Step 1
12  9 14  1 10 16  7 20
19  6 11 15 18  8 13 17
Time Step 2
12  9  1 21 10 16  7  1
19  6 11 15 18  8 13 17
Time Step 3
12  1 14 21 10 16  1 20
19  6 11  1 18  8 13 17
Time Step 4
 1  9 14 21 16  1  7 20
19  6  1 15 18  8 13  1
Time Step 5
12  9 14 21  1 16  7 20
 1  6 11 15 18  1 13 17
Time Step 6
12  9 14 21 10 16  7 20
 1  6 11 15 18  1 13 17
Time Step 7
12  9 14 21 10 16  7 20
19  6 11 15  1  8 13 17
```

1. In the Data Prompter initial dialog box, select **Grid or Scattered File**, select the leftmost button in the **Grid type** row (regular grid) and deactivate the **Single time step** toggle button.
2. Confirm that the **Block** toggle button is activated and click on **Describe Data**. The **Full prompter** dialog box appears. (Because this example *requires* the full prompter, the simplified dialog box is not accessible.)
3. Enter the path name of the data file in the **Data file** field:

   `/usr/lpp/dx/samples/data/record_series`
4. Activate the **Header** toggle button.
5. Step 4 also enables the **# of bytes** button to the right. Click on this button and select **string marker**.

6. Enter “Time Step 1 \n” in the associated text field to specify that the data file is to be read starting with the line after “Time Step 1” (see Note). Alternatively, selecting **# of lines** and specifying the value “2” in the text field would produce the same result.

   Note: The new-line character “\n” must be included in the specification, and the spacing between it and the marker must match that in the data file (e.g., if “Time Step 1” and “\n” are separated by three spaces in the file, they must be separated by three spaces in the specification). This spacing is easily determined in the Data Browser by placing the cursor at each point and reading the corresponding offset value (see Figure 18 on page 110).

7. Enter the value “5” in each of the first two **Grid size** fields.
8. Activate the **Series** toggle button and specify the number of series members by entering the value “7” in the associated **n** field. (Leave the **start** and **delta** fields unchanged.)
9. Activate the **Series separator** toggle button, select **# of lines**, and enter the value “1” in the associated text field. (When the Import module reads the data file, it will skip the lines “Time Step 2,” “Time Step 3,” and so on.)
10. Change **Dependency** from “positions” to “connections.” To confirm this change, click on **Modify** at the bottom of the dialog box (note the instruction there).
11. Repeat Example 1, Step 10 (see To save the header file on page 68), to save the header file, which should look like:

```
file = /usr/lpp/dx/samples/data/record_series
grid = 5 x 5
format = text
interleaving = record
majority = row
header = marker “Time Step 1 \n”
series = 7, 1, 1, separator = lines 1
field = field0
structure = scalar
type = float
dependency = connections
positions = regular, regular, 0, 1, 0, 1

end
```

Note: For scalar data, as in this example, the **interleaving** keyword is not required (it defaults to record). However, when series data include vectors, this keyword must be included and the appropriate value specified. For more information, see “interleaving” on page 90.

Example 9. Data and Header in the Same File

The **header** and **end** keywords make it possible to combine header information and data in the same file.

Note: Although the General Array Importer is designed to process files that contain both header information and data, the Data Prompter is not. It cannot create them or read them in. Such files, like the one in this example, must be created with an editor.

```
# The Importer disregards this line, since it is a comment line.
grid = 5 x 5
dependency = connections
type = byte
structure = scalar
format = ascii
header = marker “Start data \n”

end

# There may be comments about the data here (e.g., who created it and
# when).  These will be passed over because of the marker specified
# in the header keyword statement.

Start data
17  8  11     4    12 18  3  9
10  2  19    13     1  7 14 20
```

The **end** keyword marks the end of the header section. The **header** keyword statement specifies “Start data” as the search string and the next line as the start of the actual data. Note that if the data starts on the *same* line, the new-line character (\n) is not required as part of the marker (see also Step 6 of Example 8 in Enter “Time Step 1 \n” on page 74).

The “positions” keyword, omitted in this example, defaults to an origin of [0 0] and deltas of [1 1].

Record Style: Multivariable Data

To import record-style data, you must set the **interleaving** keyword to record, record-vector, or series-vector. (When using the Data Prompter, select Block for **Field interleaving**.) If the data includes vectors, select the appropriate vector interleaving, as discussed in “Examples 6 and 7. Vector Data” on page 71; see also “interleaving” on page 90.

Example 1. Multiple Scalar Fields

This example illustrates the importation of multiple scalar fields. The grid is 4 × 2 × 3, with an origin of [0 0 0] and deltas of [1 1 1]. The three data variables are scalar. The data file looks like:

```
Energy
    2.158719     1.45419    1.566509    1.551361    2.215095    1.726923
    2.080461    1.418617    1.373206    2.231642    1.316575    1.445211
    1.673182    1.445737    1.820333    2.167849    1.721611    1.554906
    1.604594    2.061092    1.398391    2.062042    1.996196     1.50964
Pressure
    34.81398    18.81529    29.65139      42.499    22.96053    31.41604
    19.92936    27.79935    26.34873    28.91081    21.17855    28.89354
    6.320079     43.9068    6.597938    20.41342    14.83351    43.53309
    16.36901    18.19812    4.628566    43.64742    44.99699    26.32183
Temperature
    295.3329    302.5431     301.835    296.0127    297.8344    295.5451
    301.6786    298.4496    302.0944    296.7458    296.3459    296.4179
    303.1223    300.3094    297.9714    300.0774    299.1322    296.9368
     302.096    294.8137     300.662    299.5744    304.1986    302.4216
```

The header file to import this data should look like:

```
file = /usr/lpp/dx/samples/data/record_multiscalar
grid = 4 x 2 x 3
format = text
interleaving = record
majority = row
header = lines 1
field = Energy, Pressure, Temperature
recordseparator = lines 1

end
```

Example 2. Cell-Centered Data

This example is identical to the preceding one except that each of the data variables is dependent on the connections between data points rather than on their positions. Thus there are only six data values per field (3 × 1 × 2). The data file looks like:

```
Energy
    2.158719      1.45419     1.566509     1.551361     2.215095     1.726923
Pressure
    34.81398     18.81529     29.65139       42.499     22.96053     31.41604
Temperature
    295.3329     302.5431      301.835     296.0127     297.8344     295.5451
```

The header file to import this data should look like:

```
file = /usr/lpp/dx/samples/data/record_multiscalardepconn
grid = 4 x 2 x 3
format = test
interleaving = record
majority = row
header = lines 1
field = Energy, Pressure, Temperature
recordseparator = lines 1
dependency = connections

end
```

Example 3. Multiple Scalars with Mixed Dependencies

This example differs from the preceding one in that Energy and Temperature are dependent on the positions of the grid, while Pressure is dependent on the grid elements (connection dependent). The data file looks like:

```
Energy
    2.158719      1.45419     1.566509     1.551361     2.215095     1.726923
    2.080461     1.418617     1.373206     2.231642     1.316575     1.445211
    1.673182     1.445737     1.820333     2.167849     1.721611     1.554906
    1.604594     2.061092     1.398391     2.062042     1.996196      1.50964
Pressure
    34.81398     18.81529     29.65139       42.499     22.96053     31.41604
Temperature
    295.3329     302.5431      301.835     296.0127     297.8344     295.5451
    301.6786     298.4496     302.0944     296.7458     296.3459     296.4179
    303.1223     300.3094     297.9714     300.0774     299.1322     296.9368
     302.096     294.8137      300.662     299.5744     304.1986     302.4216
```

The header file looks like the one shown at the top of the next page.

```
file = /usr/lpp/dx/samples/data/record_multiscalarmixed
grid = 4 x 2 x 3
format = text
interleaving = record
majority = row
header = lines 1
field = Energy, Pressure, Temperature
dependency = positions, connections, positions
recordseparator = lines 1

end
```

Examples 4 and 5. Scalar and Vector Data

This example uses the same grid as the previous 3, but here the second data field (velocity) consists of 2-vectors. In Example 4, all the x-components of the 2-vectors are listed first, followed by all the y-components. For example, the x- and y-components of the first 2-vector are 34.81398 and 2.158719, respectively.

```
Energy
    2.158719     1.45419    1.566509    1.551361    2.215095    1.726923
    2.080461    1.418617    1.373206    2.231642    1.316575    1.445211
    1.673182    1.445737    1.820333    2.167849    1.721611    1.554906
    1.604594    2.061092    1.398391    2.062042    1.996196    1.509640
Velocity
    34.81398    18.81529    29.65139      42.499    22.96053    31.41604
    19.92936    27.79935    26.34873    28.91081    21.17855    28.89354
    6.320079     43.9068    6.597938    20.41342    14.83351    43.53309
    16.36901    18.19812    4.628566    43.64742    44.99699    26.32183
    2.158719     1.45419    1.566509    1.551361    2.215095    1.726923
    2.080461    1.418617    1.373206    2.231642    1.316575    1.445211
    1.673182    1.445737    1.820333    2.167849    1.721611    1.554906
    1.604594    2.061092    1.398391    2.062042    1.996196    1.509640
Temperature
    295.3329    302.5431     301.835    296.0127    297.8344    295.5451
    301.6786    298.4496    302.0944    296.7458    296.3459    296.4179
    303.1223    300.3094    297.9714    300.0774    299.1322    296.9368
    302.0960    294.8137     300.662    299.5744    304.1986    302.4216
```

The header file should look like:

```
file = /usr/lpp/dx/samples/data/record_scalarvector1
grid = 4 x 2 x 3
format = text
interleaving = record
majority = row
header = lines 1
field = Energy, Velocity, Temperature
structure = scalar, 2-vector, scalar
recordseparator = lines 1, lines 0, lines 1

end
```

Note that the interleaving specified for the vectors (line 4) is record (see “interleaving” on page 90) and that the record separator (line 9) specifies: one (1) line separating the Energy and Velocity data; no lines separating the records containing the components of the Velocity data; and one (1) line separating the Velocity and the Temperature data (see “recordseparator” on page 92).

The data values in Example 5 are the same as those in Example 4, but the components of each vector in the Velocity field appear together (e.g., 34.813980 is followed by 2.158719 in the same row):

```
Energy
   2.158719   1.454190     1.566509   1.551361     2.215095   1.726923
   2.080461   1.418617     1.373206   2.231642     1.316575   1.445211
   1.673182   1.445737     1.820333   2.167849     1.721611   1.554906
   1.604594   2.061092     1.398391   2.062042     1.996196   1.509640
Velocity
  34.813980   2.158719    18.815290   1.454190    29.651390   1.566509
  42.499001   1.551361    22.960529   2.215095    31.416040   1.726923
  19.929359   2.080461    27.799351   1.418617    26.348730   1.373206
  28.910810   2.231642    21.178551   1.316575    28.893539   1.445211

   6.320079   1.673182    43.906799   1.445737     6.597938   1.820333
  20.413420   2.167849    14.833510   1.721611    43.533089   1.554906
  16.369011   1.604594    18.198120   2.061092     4.628566   1.398391
  43.647419   2.062042    44.996990   1.996196    26.321831   1.509640
Temperature
 295.332886  302.543091  301.834991  296.012695  297.834412  295.545105
 301.678589  298.449585  302.094391  296.745789  296.345886  296.417908
 303.122314  300.309387  297.971405  300.077393  299.132202  296.936798
 302.096008  294.813690  300.661987  299.574402  304.198608  302.421600
```

The header file should look like:

```
file = /usr/lpp/dx/samples/data/record_scalarvector2
grid = 4 x 2 x 3
format = text
interleaving = record-vector
majority = row
header = lines 1
structure = scalar, 2-vector, scalar
field = Energy, Velocity, Temperature
recordseparator = lines 1

end
```

Note that the interleaving specified for the vectors (line 4) has been changed to record-vector and that the record separator (line 9) specifies one (1) line separating successive records.

Example 6. Deformed (Warped) Regular Grid

A deformed regular grid (sometimes referred to as a warped grid) is one in which the positions are irregular but the connections are regular. In this example the grid is 5 × 4. The data consists of three records, the first two of which contain scalar data defined on the grid. The third contains 2-vector values defining the grid positions. The Data Prompter uses the reserved word **locations** as a field name for the x,y values of the grid positions. The data file contains no descriptive information.

1. In the Data Prompter initial dialog box, click on **Grid or Scattered File**, then on the button representing deformed data (third from the left) in the row labeled **Grid type**.
2. Set **Number of variables** to "2."
3. Step 1 automatically activates the **Positions in data file** toggle button (the positions of warped regular data are assumed to be listed in the data file) and

displays a **Dimension** stepper button. Set the **Dimension** value to “2” for 2-dimensional (x,y) data.

4. For **Data organization**, confirm that the **Block** (i.e., record) toggle button is activated.
5. Click on **Data Prompter** to bring up the simplified data prompter.
6. Enter the path name of the data file in the **Data file** field:

   `/usr/lpp/dx/samples/data/record_deformed`
7. Enter the values “5,” and “4,” in that order, in the first two **Grid size** fields.
8. Confirm that **Data format**, **Data order**, and **Vector interleaving** are set respectively to Text, Row, and record-vector (X_0Y_0, X_1Y_1, ...).
9. In the **Field list**, click on “locations” and use the “Move field” stepper arrows to position it after “field1” (i.e., at the bottom of the list). This change is necessary to reflect the actual data file, where the two scalar fields precede the x,y positions. (By default, the Data Prompter lists “locations” as the first field.)
10. Click on “field0” in the list and then change the name to “rainfall” in the **Field name** field. To confirm this change, click on **Modify** at the bottom of the dialog box (note the instruction there).
11. Now change “field1” to “temperature” and the **Type** option from “float” to “int” (the temperature values are integers). These changes can be confirmed together by clicking on **Modify** at the bottom of the panel.
12. Save the header file (Step 10 of Example 1 in To save the header file on page 68), which should look like:

```
file = /usr/lpp/dx/samples/data/record_deformed
grid = 5 x 4
format = text
interleaving = record-vector
majority = row
field = rainfall, temperature, locations
structure = scalar, scalar, 2-vector
type = float, int, float
dependency = positions, positions, positions

end
```

Example 7. Scattered Data

This example illustrating the importation of scattered data differs from Example 6 in only a few details, mainly in specifying the number of data points instead of the dimensions of a data grid.

1. In the Data Prompter initial dialog box, select **Grid or Scattered File**, then click on the rightmost button (scattered data) in the row labeled **Grid type**.
2. Set **Number of variables** to “2.”
3. Activate the **Positions in data file** toggle button and set the **Dimension** value to “2”.
4. Repeat Steps 5 through 11 of Example 6, with the following exception: in Step 7, enter the value “20” in the **# of points** field (this change from Example 6 is a result of the original selection for scattered data in the initial dialog box).

The header file should look like:

```
file = /usr/lpp/dx/samples/data/record_deformed
points = 20
format = text
interleaving = record-vector
field = rainfall, temperature, locations
structure = scalar, scalar, 2-vector
type = float, int, float
dependency = positions, positions, positions

end
```

Example 8. Using the Block Keyword

The **block** keyword is used with record-style, fixed-format ASCII data to skip information in a block of data. For example, consider the following data file:

```
row 1 temperature 39 29 33 56 32
row 2 temperature 32 33 25 33 22
row 3 temperature 31 23 41 53 19
row 4 temperature 43 59 43 21 28
row 5 temperature 23 19 35 46 32
```

1. In the Data Prompter initial dialog box, select **Grid or Scattered File**, then ensure that the leftmost button in the row labeled **Grid type** is selected.
2. Click on **Data Prompter** to bring up the simplified prompter.
3. Enter the path name of the data file in the **data file** field:

   `/usr/lpp/dx/samples/data/block_example.data`
4. Activate the **header** toggle button, reset **# of bytes** to **# of lines**, and type 1 in the associated field.
5. Set **Grid size** to 5 × 5.
6. Now click on **Full prompter** in the **Options** pull-down menu.
7. Activate the **Block** toggle button (right-hand panel).
8. Set **skip** to 17, **# elem** to 5, and **width** to 3.
9. Repeat Example 1, Step 10 (To save the header file on page 68), to save the header file, which should look like:

```
file = /usr/lpp/dx/samples/data/block_example.data
grid = 5 x 5
format = text
interleaving = record
majority = row
header = lines 1
field = field0
structure = scalar
type = int
dependency = positions
block = 17, 5, 3
positions = regular, regular, 0, 1, 0, 1

end
```

The **block** statement instructs the importer to skip 17 characters and read 5 (temperature) values (per line in this case), reading each value from a field of three characters.

Columnar Style

Importing columnar-style data requires setting the **interleaving** keyword to “field”: Activate the **Columnar** toggle button in the Data Prompter initial dialog box or select “Field” for the **Field interleaving** option in the full prompter.

Example 1. Scalar and Vector Data on a Regular Grid.

This example illustrates the importation of a data file that contains two variables (pressure and velocity) in spreadsheet style. The data are in row majority order (last index varies fastest) and organized in four columns: the first contains the pressure data; the other three, the velocity components. The grid is 5 × 8 × 6.

1. In the Data Prompter initial dialog box, select **Grid or Scattered File**, then select the leftmost button in the row labeled **Grid type** (regular grid).
2. Set **Number of variables** to “2.”
3. For **Data organization**, activate the **Columnar** toggle button.
4. Click on **Describe Data** to bring up the simplified prompter.
5. Enter the path name of the data file in the **Data file** field:

   `/usr/lpp/dx/samples/data/spreadsheet_2var`

6. Enter the values “5,” “8,” and “6,” in that order, in the first three **Grid size** fields.
7. In the **Field list**, change the name “field0” to “pressure” and confirm the change by clicking on the **Modify** button at the bottom of the panel.
8. Change the name of “field1” to “velocity” and its **Structure** to “3-vector.” (See Steps 9 through 11 of Example 6 in the preceding section, “Example 6. Deformed (Warped) Regular Grid” on page 78, for procedure.)
9. Save the header file (Step 10 of Example 1 in To save the header file on page 68), which should look like:

```
file = /usr/lpp/dx/samples/data/spreadsheet_2var
grid = 5 x 8 x 6
format = text
interleaving = field
majority = row
field = temperature, velocity
structure = scalar, 3-vector
type = float, float
dependency = positions, positions
positions = regular, regular, regular, 0, 1, 0, 1, 0, 1

end
```

Example 2. Deformed (Warped) Regular Grid

This example differs from Example 6 in the preceding section (“Example 6. Deformed (Warped) Regular Grid” on page 78) in its data style (spreadsheet), smaller data grid (5 × 4), and number of variables (1). Follow the first 7 steps of that example, except for the following:

- In Step 2, set **Number of variables** to “1.”
- In Step 4, activate the **Columnar** toggle button.
- In Step 6, use the path name:

  `/usr/lpp/dx/samples/data/spreadsheet_deformed.`

- In Step 7, use the values “3” and “4.”

The header file should look like:

```
file = /usr/lpp/dx/samples/data/spreadsheet_deformed
grid = 3 x 4
format = text
interleaving = field
majority = row
field = locations, field0
structure = 2-vector, scalar
type = float, float

end
```

Example 3. Scattered Scalar Data

This example uses the same data set as Example 2 but treats the values as scattered data points. The data file contains an x,y position followed by a data value. There are no implied connections for these data.

1. In the Data Prompter initial dialog box:
   a. Select **Grid or Scattered File**, then select for scattered data (rightmost grid button).
   b. Activate the **Position** toggle button.
   c. Set **Dimension** to "2."
   d. Activate the **Spreadsheet** toggle button.
   e. Click on **Describe Data** to bring up the simplified data prompter.
2. Enter the path name used in Example 2.
3. Enter "12" in the **# of points** field.
4. Save the header file (Step 10 of Example 1 in To save the header file on page 68), which should look like:

```
file = /usr/lpp/dx/samples/data/spreadsheet_deformed
points = 12
format = text
interleaving = field
field = locations, field0
structure = 2-vector, scalar
type = float, float

end
```

Example 4. Handling Interspersed Text

The **layout** keyword is used to specify which locations in a data file are to be read, thereby avoiding interspersed text. In the example data file shown here, there are no implied connections between data values.

1. In the Data Prompter initial dialog box:
   a. Select **Grid or Scattered File**, then select for scattered data (rightmost grid button).
   b. Activate the **Position** toggle button.
   c. Set **Number of variables** to "1."
   d. The data positions (latitude and longitude) are in the data file and are 2-dimensional: set **Dimension** to "2."
   e. Activate the **Columnar** toggle button.
   f. Click on **Describe Data** to bring up the simplified data prompter.
2. In the **Data file** field enter the path name:

   ```
   /usr/lpp/dx/samples/data/CO2fragment.lis
   ```

3. Activate the **Header** toggle button, select the “String marker” option, and enter “CO2_CONC \n” in the associated text field. (Note that the marker text is the heading for the third column of data and that the reading of data will start after the new-line character, at the point marked by “*” on the following line. See also Step 6 of Example 7 in Enter “Time Step 1 \n” on page 74.)

Note: The asterisk (*) at the beginning of the first data line and the interval scale following the data are for reference purposes only and do not appear in the actual file (see Steps 3 and 5f–j in this example).

```
 VARIABLES AND SPECIFIED RANGES
_______________________________________________________________________
 EPOCH               01-Jul-1983 00:00:00.000 31-Dec-1987 00:00:00.000
 LATITUDE                               -90.00                    90.00
 LONGITUD                              -180.00                   180.00
 CO2_CONC                             -10000.0                  10000.0

        EPOCH                 LATITUDE      LONGITUD      CO2_CONC
*01-Jul-1983 00:00:00.000        -37.95         77.53         341.4
 01-Jul-1983 00:00:00.000        -89.98        -24.80         341.0
 01-Jul-1983 00:00:00.000         -7.92        -14.42         343.4
 01-Jul-1983 00:00:00.000        -40.68        144.68        -100.0
 01-Jul-1983 00:00:00.000         19.52       -154.82         341.9
 01-Jul-1983 00:00:00.000        -14.25       -170.57         342.0
 01-Jul-1983 00:00:00.000          2.00       -157.30        -100.0
 01-Jul-1983 00:00:00.000         55.20       -162.72         335.3
 01-Jul-1983 00:00:00.000        -75.67        -27.00         341.7
 01-Jul-1983 00:00:00.000        -43.83       -172.63         341.3
 01-Jul-1983 00:00:00.000         25.67        -80.17         343.8
 01-Jul-1983 00:00:00.000         -4.67         55.17         339.1
 01-Jul-1983 00:00:00.000         13.43        144.78         344.0
 01-Jul-1983 00:00:00.000         19.53       -155.58         343.5
 01-Jul-1983 00:00:00.000         76.23       -119.33         339.8
 01-Jul-1983 00:00:00.000         40.05       -105.63         339.5
 01-Jul-1983 00:00:00.000         66.00          2.00         338.7
 01-Jul-1983 00:00:00.000        -64.92        -64.00         341.4
 01-Jul-1983 00:00:00.000         71.32       -156.60         340.1
 01-Jul-1983 00:00:00.000         17.75        -64.77         342.3
 01-Jul-1983 00:00:00.000         38.75        -27.08         341.1

|-----------skip 33-------------| width  12 | width  12 | width 10 |
```

4. Enter “21” in the **# of points** field.
5. Use the **layout** option to “skip” interspersed text:
    a. Select **Full prompter** from the **Options** menu.
    b. Select “locations” in the **Field list** of the Data Prompter.
    c. Activate the **Layout** toggle button (in the right-hand panel of the prompter).
    d. Bring up the Browser for the data file by clicking on the ellipsis button (**...**) next to the **Data file** field.
    e. Click on **Browser...** to view the data file.
    f. In the data file, position the cursor at the beginning of the first data value in the first data line (-37.95) and note the **Byte Offsets** value (counting from the start of the line). Enter this number (33) in the **skip** field of the **Layout** option.
    g. Enter the value “12” in the **width** field of the **Layout** option. Since the latitude-longitude pairs are 2-vectors, the Data Prompter will read the

   specified width *twice* in succession, once for each component. (Thus, the Data Prompter skips 33 characters, reads 12 characters, and then reads 12 more, as specified by the first **layout** settings. See the marked intervals at the bottom of the data file in skip 33 on page 83.)
   h. To confirm these changes, click on **Modify** at the bottom of the panel.
   i. Select “field0” in the **Field list** and rename it “CO2_concentration.”
   j. Enter the value “0” (zero) in the **skip** field and “10” in the **width** field. (Now the Data Prompter skips zero characters and then reads 10 characters. See the marked intervals at the bottom of the data file in skip 33 on page 83.)
   k. To confirm these changes, click on **Modify** at the bottom of the panel.
6. Save the header file (Step 10 of Example 1 in To save the header file on page 68), which should look like:

```
file = /usr/lpp/dx/samples/data/CO2fragment.lis
points = 21
format = text
interleaving = field
header = marker “CO2_CONC \n”
field = locations, CO2_concentration
structure = 2-vector, scalar
type = float, float
layout = 33, 12, 0, 10

end
```

Example 5. Time Series with Interspersed Text

The 21 lines of data in the preceding example represent a portion of a larger file (`/usr/lpp/dx/samples/data/CO2.lis`) containing a time series with 53 members.

1. In the Data Prompter initial dialog box, select **Grid or Scattered File**, then click on **Describe Data** to bring up the simplified Data Prompter.

2. Select **Open** from the **File** menu.

3. In the **Open a Data Prompter Header** dialog box, select `CO2fragment.lis.general` in the **Files** list and then click on **OK**.

4. Change the path name to:

   `/usr/lpp/dx/samples/data/CO2.lis`

5. Activate the **Series** toggle button and change the value in the **n** field from “1” to “53.” Do not change the **start** or **delta** field.

6. Save the header file (Step 10 of Example 1 in To save the header file on page 68), which should look like:

```
file = /usr/lpp/dx/samples/data/CO2.lis
points = 21
format = text
interleaving = field
header = marker "CO2_CONC \n"
series = 53, 1, 1
field = locations, CO2_concentration
structure = 2-vector, scalar
type = float, float
dependency = positions, positions
layout = 33, 12, 0, 10

end
```

Example 6. Column Majority Data

The General Array Importer assumes that the order of the data it imports is row majority (last index varies fastest). That is, on a 2-dimensional n × m grid, the order of data is:

$f(X_0,Y_0),\ f(X_0,Y_1),\ \ldots,\ f(X_0,Y_m),\ f(X_1,Y_0),\ f(X_1,Y_1),\ \ldots$

If the order of data is column majority (first index varies fastest), the order of data is:

$f(X_0,Y_0),\ f(X_1,Y_0),\ \ldots,\ f(X_n,Y_0),\ f(X_0,Y_1),\ f(X_1,\ Y_1),\ \ldots$

The General Array Importer will accept column-majority data if you select “Column” for the **`Data order`** option in the Data Prompter.

The file `/usr/lpp/dx/samples/data/temp_wind.lis.` contains data in column majority order. A header file that imports this data is:

```
file = temp_wind.lis
grid = 144 x 73
format = text
interleaving = field
majority = column
header = lines 9
field = temperature, wind_velocity
structure = scalar, 2-vector
type = float, float
dependency = positions, positions
layout = 39, 14, 0, 14
positions = regular, regular, -178.75, 2.5, 90.0, -2.5

end
```

5.3 Header File Syntax: Keyword Statements

The header of a General Array Importer file contains two or more of the keyword statements in the list that follows. The statements from **`block`** through **`type`** are listed in alphabetical order for convenient reference (they can be listed in any order in a header file). The placement of the first two statements in the list (**`file`** and **`grid|points`**) and the last two (**`positions`** and **`end`**) reflect syntactic requirements. The descriptions that follow this list occur in the same partly alphabetized order.

Notes:

1. A statement or part of a statement enclosed in brackets ([ ]) is optional.
2. Except for `positions`, no keyword statement can exceed one line in length.
3. Any line beginning with a pound sign (#) is interpreted as a comment and ignored.

file = *filename*

grid = num_x **x** num_y **x** num_z **x**...
or
points = *n*

[**block** = $skip_1$, $elements_1$, $width_1$, $skip_2$, $elements_2$, $width_2$, ..., $skip_f$, $elements_f$, $width_f$]

[**dependency** = $dependency_1$, $dependency_2$, ..., $dependency_f$]

[**field** = $name_1$, $name_2$, ..., $name_f$]

$$\left[\textbf{format} = \begin{bmatrix}\textbf{msb}\\ \textbf{lsb}\end{bmatrix}\ \begin{matrix}\textbf{ascii}\\ \textbf{text}\\ \textbf{binary}\\ \textbf{ieee}\end{matrix}\right]$$

$$\left[\textbf{header} = \begin{matrix}\textbf{bytes}\ n\\ \textbf{lines}\ n\\ \textbf{marker}\ \text{“}\mathit{string}\text{”}\end{matrix}\right]$$

$$\left[\textbf{interleaving} = \begin{matrix}\textbf{field}\\ \textbf{record}\\ \textbf{record-vector}\\ \textbf{series-vector}\end{matrix}\right]$$

[**layout** = $skip_1$, $width_1$, $skip_2$, $width_2$, ..., $skip_f$, $width_f$]

$$\left[\textbf{majority} = \begin{matrix}\textbf{row}\\ \textbf{column}\end{matrix}\right]$$

$$\left[\textbf{recordseparator} = \begin{matrix}\textbf{bytes}\ n\\ \textbf{lines}\ n\\ \textbf{marker}\ \text{“}\mathit{string}\text{”}\end{matrix}\ ,\ \begin{matrix}\textbf{bytes}\ n\\ \textbf{lines}\ n\\ \textbf{marker}\ \text{“}\mathit{string}\text{”}\end{matrix}\ , \ldots\right]$$

$$\left[\textbf{series} = t[,\ start,\ delta]\ \left[,\ \textbf{separator} = \begin{matrix}\textbf{bytes}\ n\\ \textbf{lines}\ n\\ \textbf{marker}\ \text{“}\mathit{string}\text{”}\end{matrix}\right]\right]$$

[**structure** = $structure_1$, $structure_2$, ..., $structure_f$]

[**type** = $type_1$, $type_2$, ..., $type_f$]

$$\left[\textbf{positions} = \begin{matrix}origin_1,\ delta_1,\ \ldots,\ origin_d,\ delta_d\\ positiontype_1,\ positiontype_2,\ \ldots,\ positiontype_d,\ position_1,\ position_2,\ \ldots,\ position_k\\ position_1,\ position_2,\ \ldots,\ position_g\end{matrix}\right]$$

[**end**]

file

file = *filename*

Function: Specifies the name of the file (including the path, if any) containing the data to be imported. The Importer searches the directory where the header file was found and any paths specified with the DXDATA environment variable (see the appropriate appendix in *IBM Visualization Data Explorer User's Guide* for information on environment variables).

Use: Required unless the header file contains an `end` statement. If this statement is omitted, the Importer assumes that the data are contained in the same file as the header and that they begin on the line immediately after the `end` statement or at the point specified by a `header` statement.

grid

grid = num_x **x** num_y **x** num_z **x**...

Function: Specifies the size and dimensions of the grid containing the data to be imported.

Use: Required unless the header file contains a `points` statement.

Notes:

1. The *num* parameter is an integer specifying the number of coordinate points for a particular dimension (e.g., num_x). The number of dimensions is implicitly specified by the number of such values provided. For example,

   ```
   grid = 2 x 2
   ```

   specifies a regular grid of 4 (four) points (the `x` between integers is required, but the blank spaces are optional.)
2. The `grid` keyword (in contrast to `points`) implies connections between data points. An *n*-dimensional cuboid is assumed for connections. For example, a 1-dimensional grid generates a line connecting the positions.
3. You can also specify that the number of grid elements are to be found in the data file. For the syntax, see B.1, "General Array Importer: Keyword Information from Data Files" on page 242 in *IBM Visualization Data Explorer User's Guide*.

points

points = *n*

Function: Specifies the number of data points to be imported.

Use: Required unless the header file contains a `grid` statement.

Notes:

1. The `points` keyword (unlike `grid`) implies an absence of connections between data points.
2. Unless the `locations` reserved word is used in the `field` keyword statement (see "field" on page 88), the positions are 1-dimensional. If this reserved word is not used, 1-dimensional positions can be specified with the `positions` keyword (see "positions" on page 94). Otherwise, positions are assumed to be regular, with an origin of 0 (zero) and a delta of 1 (one).
3. You can also specify that the number of points is to be found in the data file. For the syntax, see B.1, "General Array Importer: Keyword Information from Data Files" on page 242 in *IBM Visualization Data Explorer User's Guide*.

block

[**block** = *skip*$_1$, *elements*$_1$, *width*$_1$, *skip*$_2$, *elements*$_2$, *width*$_2$, ..., *skip*$_f$, *elements*$_f$, *width*$_f$]

Function: Specifies characteristics of each data field being imported. This keyword applies only to fixed-format ASCII data with record, record-vector, or series-vector interleaving.

Use: Optional.

Note: All three parameters take integer values. (Comma separators are optional.)

- `skip` specifies the number of leading characters (in a line) to be passed over before reading the data in a line or in a record.
- `elements` specifies the number of data values stored in each line or in an entire record.
- `width` specifies the number of characters to be read for each element of a given field.

dependency

[**dependency** = *dependency*$_1$, *dependency*$_2$, ..., *dependency*$_f$]

Function: Specifies the dependencies of the data fields being imported.

Use: Optional. By default, data are assumed to be position dependent. Only if the header file also contains a **`grid`** statement (see “grid” on page 87) may you specify connections (cell-centered) dependency (a **`points`** statement implies positions dependency).

Notes:

1. For field-interleaved data, all fields must have the same dependency.
2. The **`locations`** field (see “field Keyword Statement,” Note 4) must be specified as position dependent.
3. Comma separators are optional.

field

[**field** = *name*$_1$, *name*$_2$, ..., *name*$_f$]

Function: Specifies the name and number of individual fields in a data file.

Use: Optional. If this keyword is not used, the number of fields is derived from other keywords (e.g., **`structure`** or **`type`**).

Notes:

1. The values for the **`name`** parameter are strings separated by commas and without quotation marks.
2. You must name all the fields in the data file.
3. The field names can be used later to refer to individual fields. For example, if you use the Import module to import the data all at once, you can use the Select module to separate out each field by name. Or you can specify the names of the fields you want to import as the **`variable`** parameter to the Import module. For more information, see “Import” on page 165 and “Select” on page 291 in *IBM Visualization Data Explorer User’s Reference*.
4. If the position values are intermixed with the data in a file, you must specify the positions as a field (instead of using the **`positions`** keyword). Use the reserved

word **`locations`** for the field name. The corresponding value for the **`structure`** keyword should specify the dimensionality of the positions (e.g., “2-vector”).

format

$$\left[\ \textbf{format} = \begin{bmatrix}\textbf{msb}\\ \textbf{lsb}\end{bmatrix}\ \begin{matrix}\textbf{ascii}\\ \textbf{text}\\ \textbf{binary}\\ \textbf{ieee}\end{matrix}\ \right]$$

Function: Specifies the format and byte order of the data.
Use: Optional.

Notes:

1. The accepted values for byte order are **`msb`** (most significant byte first) and **`lsb`** (least significant byte first). If the format is specified as binary, then the default byte order is the host byte order (i.e., the byte order of the machine on which the Import module is executing).
2. For specifying the data format, **`ascii`** and **`text`** are synonymous, as are **`binary`** and **`ieee`**. The default is **`ascii`**. The supported binary form is IEEE.

header

$$\left[\ \textbf{header} = \begin{matrix}\textbf{bytes}\ n\\ \textbf{lines}\ n\\ \textbf{marker}\ \text{“}\textit{string}\text{”}\end{matrix}\ \right]$$

Function: Specifies how much material the Importer must skip before it begins reading data from a data file.
Use: Optional. By default, the Importer assumes that the data begin at the start of the file.

- **`bytes`** *n*: the Importer will pass over *n* bytes (in a binary file) or *n* characters (in an ASCII file) before it begins to read data.
- **`lines`** *n*: the Importer will pass over *n* lines before it begins to read data.
- **`marker`** “*string*”: the Importer will begin at the first character after the specified string. Quotation marks are required for a string containing blank spaces or commas.

Notes:

1. If the data begins on the line following a marker, be sure to specify a new-line character (“\n”) as part of the string. Note also that the spacing between the marker and the new-line character must be the same as that in the actual file. This spacing is easily determined in the Data Browser by placing the cursor at the end of the marker and then at the end of the line and reading the corresponding offset values (see Figure 18 on page 110).
2. If the marker itself contains quotation marks or special characters, use the escape character (“\”) to indicate them, as shown in the following table.

backslash	\	\\
backspace	BS	\b
bit pattern	ddd	ddd
carriage return	CR	\r
double quote	"	\"
form feed	FF	\f
horizontal tab	HT	\t
newline	NL (LF)	\n

Note: An octal value (ddd) can be used to specify special characters other than those shown here.

interleaving

interleaving =
field
record
record-vector
series-vector

Function: Specifies to the Importer how the data in a data file are interleaved.

Use: Optional. By default, the Importer assumes that the interleaving is record-vector.

Note: The examples presented here are based on a 1-dimensional grid with 10 elements and two series members:

```
⋮
grid = 10
series = 2
field = t, v
structure = scalar, 3-vector
⋮
```

where **t** is a scalar value and **v** is a vector with three components (*vx, vy,* and *vz*). In the examples themselves, **s** represents a series member (0 or 1), and **g** represents a grid element number (0 through 9). The **interleaving** options are as follows:

field

Specifies column-oriented data such as that generated by a spreadsheet or by data-listing software. There is a separate “column” for each of the two fields with one element of each field per “row.” (For non-scalar data, as here, there is a column for each vector component.) The number of rows corresponds to the size of the grid multiplied by the number of series members.

```
|- Field t -|--------------- Field v ---------------|

t(s=0,g=0),  vx(s=0,g=0),  vy(s=0,g=0),  vz(s=0,g=0),
t(s=0,g=1),  vx(s=0,g=1),  vy(s=0,g=1),  vz(s=0,g=1),
⋮
t(s=0,g=9),  vx(s=0,g=9),  vy(s=0,g=9),  vz(s=0,g=9),
t(s=1,g=0),  vx(s=1,g=0),  vy(s=1,g=0),  vz(s=1,g=0),
t(s=1,g=1),  vx(s=1,g=1),  vy(s=1,g=1),  vz(s=1,g=1),
⋮
t(s=1,g=9),  vx(s=1,g=9),  vy(s=1,g=9),  vz(s=1,g=9)
```

record

Specifies block or record-oriented data, where the values of all the elements of all the fields corresponding to one member (e.g., a time step) are listed before the elements and fields of the next member. For non-scalar fields, all the values of each vector component (e.g., all values of x) are listed in a separate record rather than in tuples (as they are in **record-vector** data; see below).

Note: For scalar fields, **record** and **record-vector** are the same.

```
t(s=0,g=0), t(s=0,g=1), ..., t(s=0,g=9),    ] Field t ┐
vx(s=0,g=0), vx(s=0,g=1), ..., vx(s=0,g=9), ┐         │ Member
vy(s=0,g=0), vy(s=0,g=1), ..., vy(s=0,g=9), │ Field v │   s0
vz(s=0,g=0), vz(s=0,g=1), ..., vz(s=0,g=9), ┘         ┘
t(s=1,g=0), t(s=1,g=1), ..., t(s=1,g=9),    ] Field t ┐
vx(s=1,g=0), vx(s=1,g=1), ..., vx(s=1,g=9), ┐         │ Member
vy(s=1,g=0), vy(s=1,g=1), ..., vy(s=1,g=9), │ Field v │   s1.
vz(s=1,g=0), vz(s=1,g=1), ..., vz(s=1,g=9)  ┘         ┘
```

The remaining two options (**record-** and **series-vector**) apply to cases in which vector components are stored together:

record-vector

Here the values of all the elements of all the components of all the fields corresponding to each member (e.g., a time step) are listed before those corresponding to the next member (e.g., all the data for $\mathbf{s}_0$ are listed first, followed by all the data for $\mathbf{s}_1$).

In addition, the components of each vector are stored as tuples (in contrast to the way they are stored in **record** data).

```
t(s=0,g=0), t(s=0,g=1), ..., t(s=0,g=9), ] Field t ┐
vx(s=0,g=0), vy(s=0,g=0), vz(s=0,g=0),   ┐          │
vx(s=0,g=1), vy(s=0,g=1), vz(s=0,g=1),   │ Field    │  Member s0
...                                      │   v      │
vx(s=0,g=9), vy(s=0,g=9), vz(s=0,g=9),   ┘          ┘
t(s=1,g=0), t(s=1,g=1), ..., t(s=1,g=9), ] Field t ┐
vx(s=1,g=0), vy(s=1,g=0), vz(s=1,g=0),   ┐          │
vx(s=1,g=1), vy(s=1,g=1), vz(s=1,g=1),   │ Field    │  Member s1
...                                      │   v      │
vx(s=1,g=9), vy(s=1,g=9), vz(s=1,g=9)    ┘          ┘
```

series-vector

Here the values of all the elements of all the members (e.g., time steps) are listed for one field before those of the next field (e.g., all the data for field **t** are listed first, followed by all the data for field **v**).

In addition, the components of a vector are stored as tuples rather than in separate records (as they are stored in **record** data; see above).

```
t(s=0,g=0), t(s=0,g=1), ..., t(s=0,g=9),  ┐ Field  ] Member s0
t(s=1,g=0), t(s=1,g=1), ..., t(s=1,g=9),  ┘   t    ] Member s1
vx(s=0,g=0), vy(s=0,g=0), vz(s=0,g=0),    ┐        ┐
vx(s=0,g=1), vy(s=0,g=1), vz(s=0,g=1),    │        │ Member
...                                       │        │   s0
vx(s=0,g=9), vy(s=0,g=9), vz(s=0,g=9),    │ Field  ┘
vx(s=1,g=0), vy(s=1,g=0), vz(s=1,g=0),    │   v    ┐
vx(s=1,g=1), vy(s=1,g=1), vz(s=1,g=1),    │        │ Member
...                                       │        │   s1
vx(s=1,g=9), vy(s=1,g=9), vz(s=1,g=9)     ┘        ┘
```

layout

[**layout** = $skip_1$, $width_1$, $skip_2$, $width_2$, ..., $skip_f$, $width_f$]

where *f* is the number of fields.

Function: Specifies the number of bytes (characters) the Importer must skip before it begins to read a field's data and then the number of bytes it should read (i.e., the "width" of the data item).

Use: Optional. This keyword applies only to ASCII, field-interleaved data. If the data is in ASCII format but the keyword is not used, the Importer assumes a default of one or more blank spaces (space, tab, new line, or form feed) as the delimiter between fields.

Notes:

1. The components of a vector must each be represented in the same number of characters. Comma separators are optional.

   The following statement tells the Importer to skip 10 characters, read one field of 6 characters, skip 10 characters, and read another field of 4 characters.

   ```
   layout = 10, 6, 10, 4
   ```

2. Because *skip* specifies the number of characters to be passed over before each *field*, it does not apply to single elements of a vector field but to the field as a whole. However, *width* applies to *each* element of the vector. For example, the following statement tells the Importer to skip 10 characters, then read a field 6 characters long:

   ```
   layout = 10, 6
   ```

   If the field *v* is a 2-vector, then each component (v_x and v_y) is 6 characters long, for a total width of 12 characters.
3. If there are no spaces between two fields, specify the skip value as 0 (zero).

majority

[**majority** = **row**
column]

Function: Specifies the organization of multidimensional arrays composing a data field.

Use: Optional. The default is **row** (last dimension varies fastest, as in the C programming language). Column majority means that the first dimension varies fastest, as in the FORTRAN programming language.

Note: The maximum number of dimensions supported for column majority is 4.

recordseparator

[**bytes** *n* , **bytes** *n*
recordseparator = **lines** *n* , **lines** *n* ...
marker "*string*" , **marker** "*string*"]

Function: Specifies the separation between records.

Use: Optional. This keyword applies only to record and record-vector interleaving.

Notes:

1. With differences to be noted here, the specification of separation is very much like that of the **`header`** keyword (see “header” on page 89).
2. If all records are separated by the same amount, a single separator value should be specified. For example, if each pair of successive records is separated by two lines, then

   ```
   recordseparator = lines 2
   ```

3. If the records are separated by different amounts, a value must be specified for each of the separators:
   - For record-vector data the number of separator values must equal the number of *fields* minus one. For example, if

     ```
     structure = scalar, scalar, 2-vector
     ```

     then two separator values must be specified: one between the two scalar fields and one between the second scalar field and the vector field.
   - For record data, the number of separator values must equal the number of *dimensions* minus one. For the preceding example, three separator values must be specified: the first between the two scalar fields; the second between the second scalar and the first vector component; and the third between the two vector components.

series

$$\left[\textbf{series} = t, [\mathit{start}, \mathit{delta}] \left[\,,\ \textbf{separator} = \begin{array}{l}\textbf{bytes}\ n\\ \textbf{lines}\ n\\ \textbf{marker}\ \text{“}\mathit{string}\text{”}\end{array}\right]\right]$$

Function: Specifies, to the Importer, information about a series.

Use: Optional. This keyword is required only for the importation of series data. The default assumes no separation between series sections. Descriptive information between series sections is described with the **`separator`** parameter. If descriptive information precedes the *first* member of the series, it can be skipped by means of a **`header`** statement.

- **`t`** is a required parameter that specifies the number of series elements in the data file. Its default value is 1.
- **`start`** and **`delta`** are optional parameters that specify the series positions. The position values are defined as:

  $\mathit{start}, \mathit{start} + \mathit{delta}, \mathit{start} + (2 \times \mathit{delta}), ..., \mathit{start} + (t - 1) \times \mathit{delta}$

  The defaults for **`start`** and **`delta`** are 0 (zero) and 1 (one) respectively.
- The following example specifies that there are four series members, with positions 0.8, 1.2, 1.6, and 2.0.

  ```
  series = 4, 0.8, 0.4
  ```

- The specification of **`separator`** is very much like that of the **`header`** keyword (see “header” on page 89).

structure

[**structure** = *structure*$_1$, *structure*$_2$, ..., *structure*$_f$]

Function: Specifies the structure of each field in a data file.

Use: Optional. The default is **`scalar`**. Accepted values are **`scalar`**, **`string[n]`**, and **`2-vector,...,`** **`9-vector`** for each field. However, **`5-vector, ...,`** **`9-vector`** cannot be specified for column-majority arrays. In **`string[n]`**, *n* specifies the length of the longest string.

Notes:

1. Use of this keyword requires specifying the structure of all fields in the file.
2. The following example specifies that the first two fields have a scalar structure, while the third is a vector with three components:

   ```
   structure = scalar, scalar, 3-vector
   ```

3. Since the default is scalar, the statement is not required if, say, six scalar fields are to be imported. But if one of these fields is vector, the statement is required; for example:

   ```
   structure = scalar, scalar, 3-vector, scalar, scalar, scalar
   ```

4. If string data contain embedded blanks, you must use the **`layout`** or **`block`** keyword to specify how the string is to be read.

type

[**type** = *type*$_1$, *type*$_2$, ..., *type*$_f$]

Function: Specifies the data type for each specified field.

Use: Optional. The default is **`float`**. The accepted values are:

`double`
`float`
`string`
`byte`
`signed byte`
`unsigned byte`
`int`
`signed int`
`unsigned int`
`short`
`signed short`
`unsigned short`

Notes:

1. Use of this keyword requires specifying the type for all fields in the file.
2. The following are pairs of equivalent types:
   - **`byte`** and **`unsigned byte`**.
   - **`short`** and **`signed short`**.
   - **`int`** and **`signed int`**.

positions

positions =
origin$_1$, *delta*$_1$, ..., *origin*$_d$, *delta*$_d$
positiontype$_1$, *positiontype*$_2$, ..., *positiontype*$_d$, *position*$_1$, *position*$_2$, ..., *position*$_k$
position$_1$, *position*$_2$, ..., *position*$_g$

Each syntax line in this diagram is discussed separately in Note 4 below.

Function: Defines the positions component of the fields in a data file.

Use: Optional. The default is regular positions in compact notation, where the origin and delta in each dimension are 0.0 and 1.0 respectively, unless the **`locations`** keyword has been used.

Notes:

1. This keyword must be placed at the end of the header file or immediately preceding the **`end`** keyword.
2. The numbers specified in a **`positions`** statement can span any number of uninterrupted lines (including carriage returns, which is not true of other keyword statements).
3. Positions can also be specified with the **`field`** statement and the **`locations`** reserved word (see "field" on page 88). This alternative is important in cases where:
    - The positions are irregular and can be interleaved with data themselves.
    - The positions vary with a series member.
    - The positions are not stored in row-majority order in ASCII.
4. There are four ways to specify positions with the **`positions`** keyword:

- **`Regular positions:`**

  positions = $origin_1$, $delta_1$, ..., $origin_d$, $delta_d$

  Specifies a regular grid, using the origin of a dimension and the spacing (delta) of the positions in that dimension. The dimensions must be specified in the same order as that in the **`grid`** keyword statement (see "grid" on page 87).

  The statement

  ```
  positions = 0.0, 1.0, 0.0, 0.5, 0.0, 1.5
  ```

  specifies the origin-delta pairs of a 6 × 6 × 3 grid, with origins of 0.0 in all three dimensions and deltas of 1.0, 0.5, and 1.5.

- **`Partially regular positions:`**

  positions = $positiontype_1$, $positiontype_2$, ..., $positiontype_d$, $position_1$, $position_2$, ..., $position_k$

  Specifies an array (regular or irregular) for each dimension from which a product is to be formed. For example, one array with the values 1, 2, 3, and a second with the values 1, 4, 8 will generate the following set of positions:

  (1, 1) (1, 4) (1, 8)
  (2, 1) (2, 4) (2, 8)
  (3, 1) (3, 4) (3, 8)

  For each dimension, specify a string indicating the type of positions for the dimension—accepted values are **`regular`** and **`irregular`**. Specify the strings for all dimensions first, then follow with the position values.

  For compact specification, the position values are two numbers: an origin and the delta value for the spacing. Specifying irregular positions requires an explicit listing of each position value, with the same number of positions as specified by the **`grid`** keyword statement. The order in which positions are specified must correspond to the order in which dimensions are specified in that statement.

  The following example specifies the positions for a 6 × 6 × 3 grid, where the first two dimensions are regular, and the third is irregular:

  ```
  position = regular, regular, irregular, 0.0, 1.0, 0.0, 0.5, 0.0, 0.5, 1.5
  ```

  Here the first dimension is regular, with positions 0, 1, 2, 3, 4, 5.
  The second dimension is regular, with positions 0.0, 0.5, 1.0, 1.5, 2.0, 2.5.
  The third dimension is irregular, with positions 0, 0.5, 1.5.

  The first few positions are (0, 0, 0) (0, .5, .5) (0, .5, 1.5) (1, 0, 0)....

- **`Completely irregular positions:`**

  positions = $position_1$, $position_2$, ..., $position_g$

  Specifies fully irregular positions: you must list all the position values.

  Note: The requirement that all positions must be listed is what distinguishes a "completely irregular" from a "partially regular" grid (discussed above). For example, the keyword statement

  `positions` `= irregular, irregular, irregular,...`

  still defines a *partially* regular grid, even though each array specified is irregular.

  The number of values you provide corresponds to the product of all the dimension specifications (i.e., the *num* values) in the **`grid`** keyword statement. The position values must be listed in row majority order, and they can be delimited by commas.

  You must specify *g* numbers, where *g* is the product of the *num* values of the dimensions all multiplied together, along with the number of dimensions (e.g., $m{\times}n{\times}o{\times}d$, where *m*, *n*, and *o* are the grid dimensions, or *num* values, and *d* is the number of dimensions).

  The following example specifies the six positions for an irregular 2 × 3 grid:

  ```
  positions = 0, 0, 0, 2, 0, 6, 2, 1, 5, 4, 7, 7
  ```

  The first and last positions are (0, 0) and (7, 7) respectively.

- **`Position information from the data file`**

  You can specify that the information for the **`positions`** keyword is to be found in the data file. For the syntax, see B.1, "General Array Importer: Keyword Information from Data Files" on page 242 in *IBM Visualization Data Explorer User's Guide*.

  Note: The **`positions`** keyword may give a compact encoding of the position. In that respect, this function differs from the **`locations`** reserved word when used with the **`field`** keyword (see "field" on page 88).

end

[**end**]

Function: Causes the Importer to stop processing header statements.
Use: Optional unless the data are in the same file with the header statements. By default, the Importer stops processing at the end of the header file.

5.4 Data Prompter

The Data Explorer Data Prompter is a stand-alone, Motif-based user interface for importing data. It consists of three dialog boxes:

- Initial Dialog Box (see Figure 15 on page 99)
- Simplified Data Prompter (see Figure 16 on page 102)
- Full Data Prompter (see Figure 17 on page 105).

The Data Prompter imports a variety of different formats. It also gives you access to some general purpose visualization programs which can visualize a wide variety of different types of data (e.g. two-dimensional, three-dimensional, scalar, vector, series, etc.)

For Future Reference

The general purpose programs used by the Data Prompter may be found in `/usr/lpp/dx/ui`, with names denoting the type of data they visualize. You may find these programs useful on their own (apart from their use in the Data Prompter).

Supported Formats

Data Explorer Format

The Data Explorer format can be used to describe any object which can be represented in Data Explorer. Objects can be exported in the Data Explorer format using the Export module, and often filters are written to convert from other formats to the Data Explorer format. The Data Explorer format is described in detail in B.2, “Data Explorer Native Files” on page 244 in *IBM Visualization Data Explorer User’s Guide*.

The Data Explorer format is supported directly by the Import module (see “Import” on page 165 *IBM Visualization Data Explorer User’s Reference*). To create visual programs using data in this format, simply use the Import module, specifying the file name, and the format as “dx”.

CDF Format

CDF is a standard format, supported directly by the Import module. For more information on the CDF format, see B.3, “CDF Files” on page 279 in *IBM Visualization Data Explorer User’s Guide*. To create visual programs using data in this format simply use the Import module, specifying the cdf as the **name** parameter to Import, and specifying the format as “cdf”.

netCDF Format

netCDF is a standard format, supported directly by the Import module. For more information on the netCDF format, see B.4, “netCDF Files” on page 281 in *IBM Visualization Data Explorer User’s Guide* To create visual programs using data in this format simply use the Import module, specifying the file name as the **name** parameter to Import, and specifying the format as “netCDF”.

HDF Format

HDF is a standard format, supported directly by the Import module. Data Explorer supports HDF files that contain a Scientific Dataset (SDS). For more information on the HDF format, see B.6, “HDF Files” on page 288 in *IBM Visualization Data Explorer User’s Guide*. To create visual programs using data in this format simply use the Import module, specifying the file name as the **name** parameter to Import, and specifying the format as “hdf”.

Image data

Images in TIFF, MIFF, GIF, and RGB formats can be directly imported by the ReadImage module (see “ReadImage” on page 250 in *IBM Visualization Data Explorer User’s Reference*). To see the image, you need only to attach the output of ReadImage to first input of the Display module. You can of course manipulate the image with any of the appropriate Data Explorer modules.

Grid or Scattered Data (General Array format)
Data Explorer can import a wide variety of gridded and scattered data using the General Array format. The basic procedure is to create a header file which describes the structure of the data (dimensionality, number of variables, layout in the file, etc.). The General Array Importer is described in detail in 5.1, "General Array Importer" on page 63. 5.2, "Importing Data: Header File Examples" on page 67 contains many examples illustrating the wide variety of data that can be imported.

The Data Prompter greatly simplifies the task of creating a header file, as it performs extensive error checking (disallowing conflicting keywords, for example) and frees you from needing to know the exact syntax of the General Array format. When you use the Data Prompter to import this format, you will be asked to describe your data in detail. You need to then save the header file using **Save As** in the File menu of the Data Prompter Full or Simplified window. The data can then be visualized using one of the general purpose programs provided by the Data Prompter.

To create new visual programs using data imported in this way, simply specify the name of the header file to the Import module, specifying the format as "general" (see "Import" on page 165 in *IBM Visualization Data Explorer User's Reference*).

Spreadsheet Data Spreadsheet data is typically non-spatial data, arranged in columns. This type of data is supported by the ImportSpreadsheet module (see "ImportSpreadsheet" on page 170 in *IBM Visualization Data Explorer User's Reference*).

Note: For the formats directly supported by Import (Data Explorer, CDF, netCDF, HDF) or ImportSpreadsheet (spreadsheet data), it is not necessary to use the Data Prompter to import the data. You can simply use the Import or ImportSpreadsheet module and then add whatever visualization modules you want to look at the data. However, you can use the Data Prompter to give you easy access to the general purpose programs which get you "up and running" with a picture of your data.

If you are importing your data using the General Array format, once you have created a header file (typically done by using the Data Prompter), you can import the header file directly using the Import module.

Initial Dialog Box

To start the Data Prompter, type:

```
dx -prompter
```

or choose **Import Data** from the Data Explorer Startup window. The initial dialog box appears (see Figure 15 on page 99).

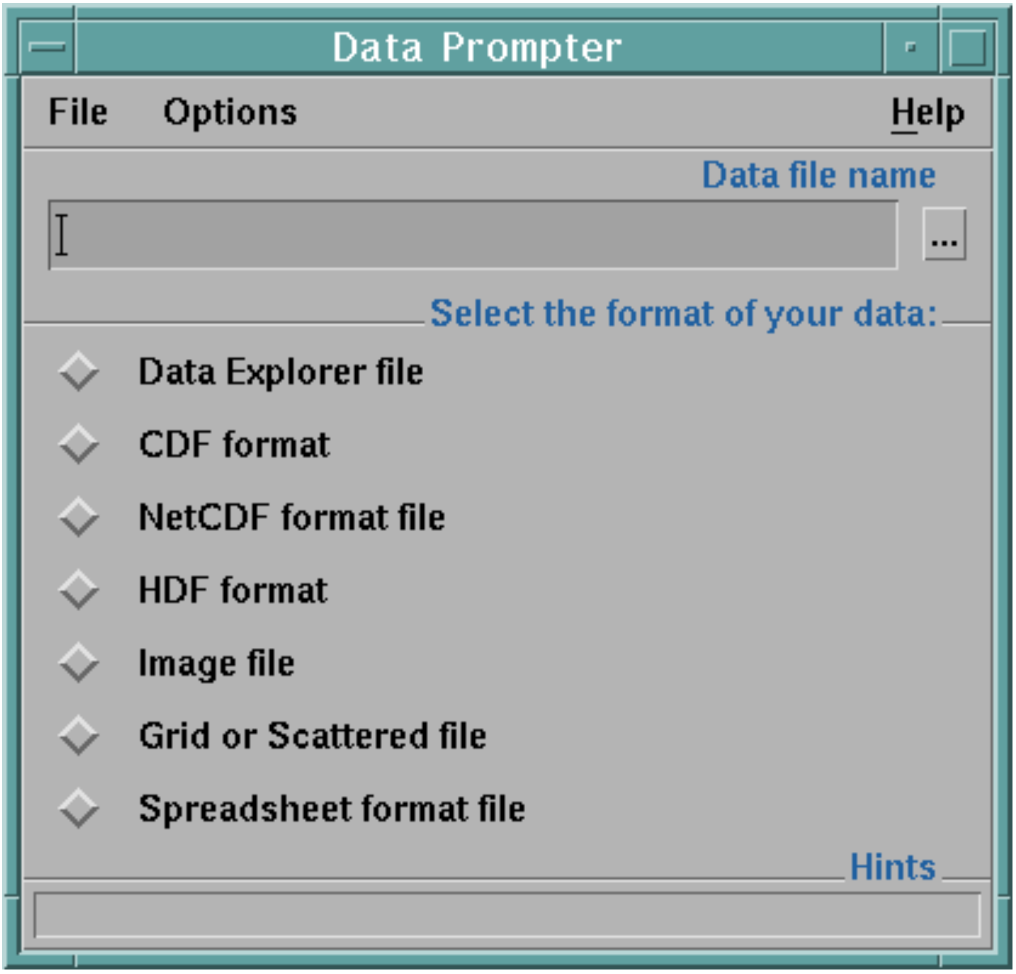

Data Prompter
File
Options
Help
Data file name
...
Select the format of your data:
◇ Data Explorer file
◇ CDF format
◇ NetCDF format file
◇ HDF format
◇ Image file
◇ Grid or Scattered file
◇ Spreadsheet format file
Hints

Figure 15. Initial Data Prompter window

At the top of the Data Prompter initial dialog is a text field into which you can enter the data file to be imported. If you press the **...** button to the right of the field, a list of directories is presented. This list of directories is taken from your `DXDATA` environment variable, if set (see C.1, “Environment Variables” on page 292 in *IBM Visualization Data Explorer User’s Guide*). Then a file selection dialog is presented, initialized to the directory chosen from the **...** button. You can also access the file selection dialog directly by choosing **Select Data File** from the **File** menu.

The dialog also allows you to specify the type of data file you wish to import. The choices are:

- Data Explorer file
- CDF format
- netCDF format file
- HDF format
- Image file
- Grid or Scattered file
- Spreadsheet format

For the Data Explorer, CDF, netCDF, and HDF formats, all you need to do is make the selection of the format and specify the file name at the top of the dialog. Each of these formats is supported directly by Data Explorer and no further description by you is necessary. Then you can choose to browse the data file, have Data Explorer test the import of the data file and print some characteristics of it, or have Data Explorer visualize the data automatically using a general purpose visual program.

If you choose `Image file`, the dialog will expand, allowing you to choose the type of image format, and you can then have Data Explorer automatically read and display your image file.

If you choose `Spreadsheet format`, the dialog will expand, allowing you to specify whether to import the data as a table or a matrix. Spreadsheet data consists of columns of related data, typically non-spatial. If imported as a table, then the data will be treated as a single Field of data, with each column placed in the field as a named component. Each component will contain scalar or string data. If imported as a matrix, then it is implicitly assumed to be a two-dimensional grid, with the rows and columns specifying the two dimensions. In this case the data in each column must be of the same type (i.e. you cannot have mixed strings and numbers). The `Spreadsheet format` option also allows you to specify a column delimiter. For example, to specify tab-separated columns, specify “\t” as the delimiter. See “ImportSpreadsheet” on page 170 in *IBM Visualization Data Explorer User’s Reference* for more information.

If you choose `Grid or Scattered File (General Array Format)`, which allows you to import a wide variety of data formats, you will need to tell Data Explorer more about the file. If you choose this option, the dialog box that appears allows you to identify for the Data Prompter five important characteristics of the data you want to import:

- Grid type
- Number of variables
- Positions in data file
- Single time step
- Data organization.

Grid type

Four grid-selection buttons display patterns representing different types of data. Reading from left to right, they are:

- *Regular grid*: the data-point positions can be specified by origin-and-delta pairs (one pair for each dimension).
- *Partly regular grid*: one or more of the dimensions cannot be described by a simple origin-delta pair.
- *Warped regular grid*: each position must be explicitly specified, but there is still a grid structure to the connections between data points.
- *Scattered data*: there are no connections between data points.

Number of variables

The stepper button allows you to specify the number of variables in the data file to be imported. For example, if the file contains data values for temperature and velocity (and for nothing else), the stepper button should be set to 2. (The default is 1, and the allowed range is 1–100.)

Positions in data file

This option is available only with the selection of warped regular grid or scattered points. It is not meant for files that “describe” data positions by reference to origin-delta pairs. For example, if the data are organized as:

```
x1, y1, data1
x2, y2, data2
. . .
```

the toggle should be activated.

Selecting the regular or partly regular grid automatically deactivates the toggle: the label is grayed out, and the button cannot be activated.

Selecting the third button (warped grid) automatically activates the toggle, and a stepper button appears for setting the number of dimensions (e.g., set the stepper to 3 for x,y,z points). Once the warped grid is selected, however, the toggle cannot be reset (i.e., the position specifications are assumed to be in the file).

Selecting the fourth button (scattered points) does not activate the toggle, but it allows you to do so, and to deactivate it even after setting the number of dimensions.

Single time step

This toggle button allows you to specify whether your data consists of a single time step or not. By default, the toggle is activated.

Data organization

The data organization can be characterized as block or spreadsheet. For example, given a regular grid containing two variables (say temperature and pressure), block style lists one set of values first, followed by the other:

$t_1, t_2, \ldots, t_n, p_1, p_2, \ldots, p_n$

Spreadsheet style alternates the two (one pair per line):

t_1, p_1
t_2, p_2
...
t_n, p_n

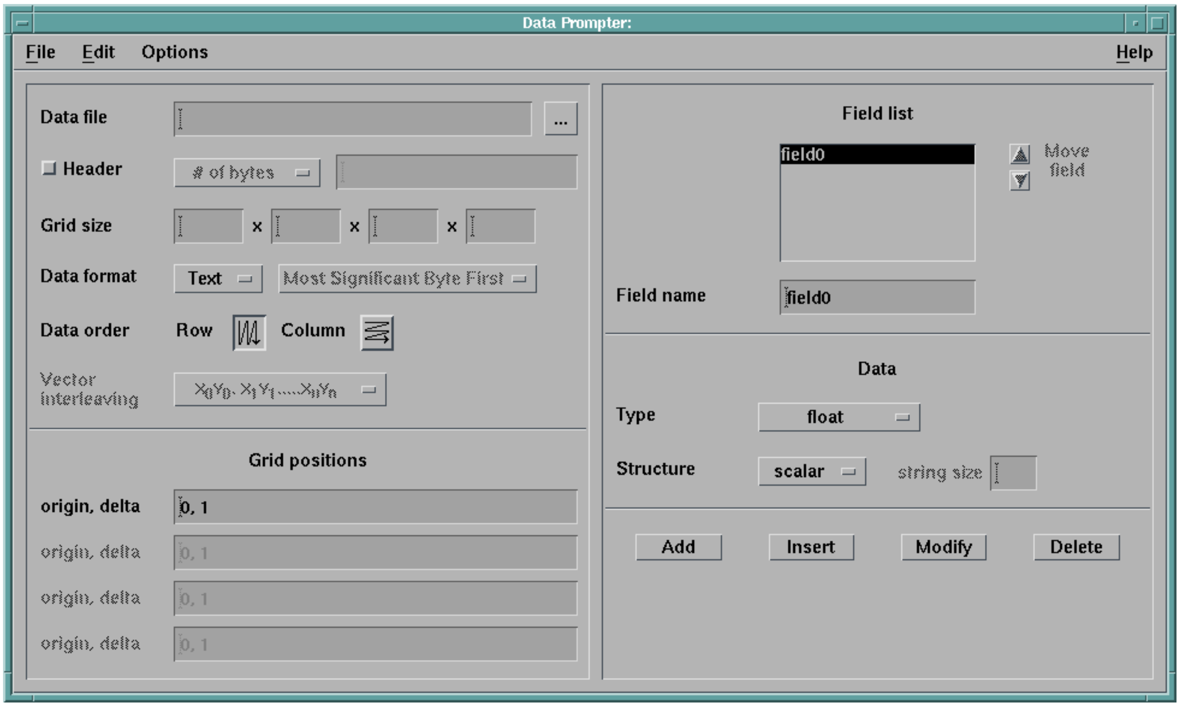
Once you have specified the five characteristics in the initial dialog box, click on **`Data Prompter`**. The simplified Data Prompter that appears is “customized,” containing only those options appropriate to the data you have described in the dialog box (see “Simplified Data Prompter”).

For Future Reference: Once you have opened and modified either the simplified or full data prompter, if you then close the simplified or full window, you should not use the **`Describe Data`** button to reopen the window, as the **`Describe Data`** button opens a brand new window. Instead, use **`Open General Array Importer`** from the **`Options`** menu of the initial dialog.

Simplified Data Prompter

The simplified Data Prompter (Figure 16 on page 102) is just what its name implies: a smaller version of the full Data Prompter (Figure 17 on page 105). It displays a subset of the buttons, parameters, and fields contained in the larger version. Since these are identical in both versions, they are described in the section on the full Data Prompter (see “Full Data Prompter” on page 103).

This section instead describes only the menu bar and its options, which are also identical in both the simplified and full prompters.



Data file
☐ Header # of bytes
Grid size: × × ×
Data format: Text, Most Significant Byte First
Data order: Row, Column
Vector interleaving: $X_0Y_0, X_1Y_1, \ldots, X_nY_n$
origin, delta: 0, 1
origin, delta: 0, 1
origin, delta: 0, 1
origin, delta: 0, 1
Field list: field0
Field name: field0
Type: float
Structure: scalar
string size

Figure 16. Simplified Data Prompter. This window contains a subset of the buttons, parameter, and fields available in the full Data Prompter (see Figure 17 on page 105). The Options pull-down menu can be used to call up the full prompter or to return to the initial dialog box.

The Data Prompter menu bar displays four options: **File**, **Edit**, **Options**, and **Help**.

Data Prompter File Pull-down Menu

The following functions are available in this pull-down menu:

New Resets the Data Prompter by setting all fields to their default values.

Open...
Invokes a standard Motif file-selection dialog box that prompts for a choice of file. You can read in an existing General Array Importer header file, whether or not it was created with the Data Prompter. However, the Data Prompter does *not* support header files containing the data to be imported: the data is assumed to exist in a separate file. To read an existing header from a file that also contains the data to be visualized, the header should be written out to a separate file.

Save Writes the Data Prompter header file out, using the current file name. The current file name is set by opening a file or by executing the **Save As...** command. The Data Prompter will check the correctness of most aspects of the header file to be saved. Any problems are reported and the **Save** operation is terminated. You may then correct the indicated problem and save the header file again.

Save As...
Is the same as **Save** except that you must specify a name for the header file.

Quit Terminates the Data Prompter application. It also gives you the option of saving any changes not already saved.

Data Prompter Edit Pull-down Menu

`Edit` has one option: the **`Comment`** dialog box.

Comment...
: Displays a text dialog box for entering comments about a header file. These comments are stored with the file and are ignored by the General Array Importer. Any comments it contains are displayed in the **`Header Comment`** dialog box.

Data Prompter Options Pull-down Menu

The following options are available in this pull-down menu:

Full prompter
: Invokes the most detailed prompter dialog box (Figure 17 on page 105). It can also be invoked from the command line: `dx -prompter -full`.

Simplified prompter
: Invokes a prompter dialog box (Figure 16 on page 102) that is less detailed than the full Data Prompter.

Initial dialog
: Invokes the initial dialog box (Figure 15 on page 99). It can also be invoked from the command line: `dx -prompter`.

Data Prompter Help Pull-down Menu

This pull-down menu contains one option not available in the corresponding pull-down menus of the VPE window, the Image window, and the Data Browser:

On General Array Format...
: Displays the online documentation of the General Array Importer format.

For details of the other menu options, see *IBM Visualization Data Explorer User's Guide*.

Full Data Prompter

The full Data Prompter dialog box is divided into halves, the left half describing features of both the data file and the data, and the right half describing the data field(s) (see Figure 17 on page 105). The order of descriptions in this section follows that of the dialog box: from top to bottom in the left half and then top to bottom in the right half.

Note: If you used the initial dialog box to describe your data and then selected **`OK`** instead of **`Full`**, you are in the simplified Data Prompter and some options may not be presented. However, you can invoke the full prompter at any time by selecting it in the **`Options`** pull-down menu.

Data File and Data Information

Data file
: The first information the General Array Importer requires is the path name of the data file to be imported. This name can be entered directly in the text field to the right of the **`Data file`** label. Alternatively, you can click on the ellipsis button (**`...`**) to the right of the text field. If you select **`File Selection Dialog...`** from the pop-up menu, you can select a file from the dialog list. Note that using the **`File Selection`** dialog list is simply a shortcut for typing in the path name. The **`Browser`** option on the ellipsis pop-up is discussed in 5.5, "Data Prompter Browser" on page 109.

See also “file” on page 86.

Header

If your data file has a header (an initial section that must not be imported with the data), activate the **Header** toggle button. Clicking on the button immediately to the right of the toggle will generate a pop-up menu of three options for specifying where the header stops and the data begins:

- **# of bytes** (from the beginning of the file)
- **# of lines** (from the top of the file)
- **string marker**

Click on one and enter the appropriate information in the associated text field to the right. The appropriate offset or string can be determined by browsing the data file. (See Notes in “header” on page 89. As earlier examples demonstrate, this information ultimately appears in a **header** keyword statement in a header file.)

See also “header” on page 89.

Grid Size versus # of Points

Located below the **Header** toggle button are two buttons labeled **Grid Size** and **# of Points**.

If the data has connections that are gridded, click on the **Grid size** button. Specify the dimensions of the grid in the associated text fields (e.g., if the data are 2-dimensional, enter the sizes in the first two fields and leave the last two blank).

If the data consist of unconnected points, click on the **# of Points** associated button, and enter the appropriate number in the associated text field.

Notice that these choices affect other aspects of the Data Prompter. The choice of points deactivates the **Data order** buttons and all but the first origin-delta field in the **Positions** section at the bottom.

See also “grid” on page 87 and “points” on page 87.

Data Format

The format of the data can be ASCII/Text or Binary/IEEE. The selection of **Binary/IEEE**) activates the **Significant Byte First** option button immediately to the right. You may then select Most Significant Byte First (MSB) or Least Significant Byte First (LSB). For ASCII (or Text) format, this option button is inactivated.

See also “format” on page 89.

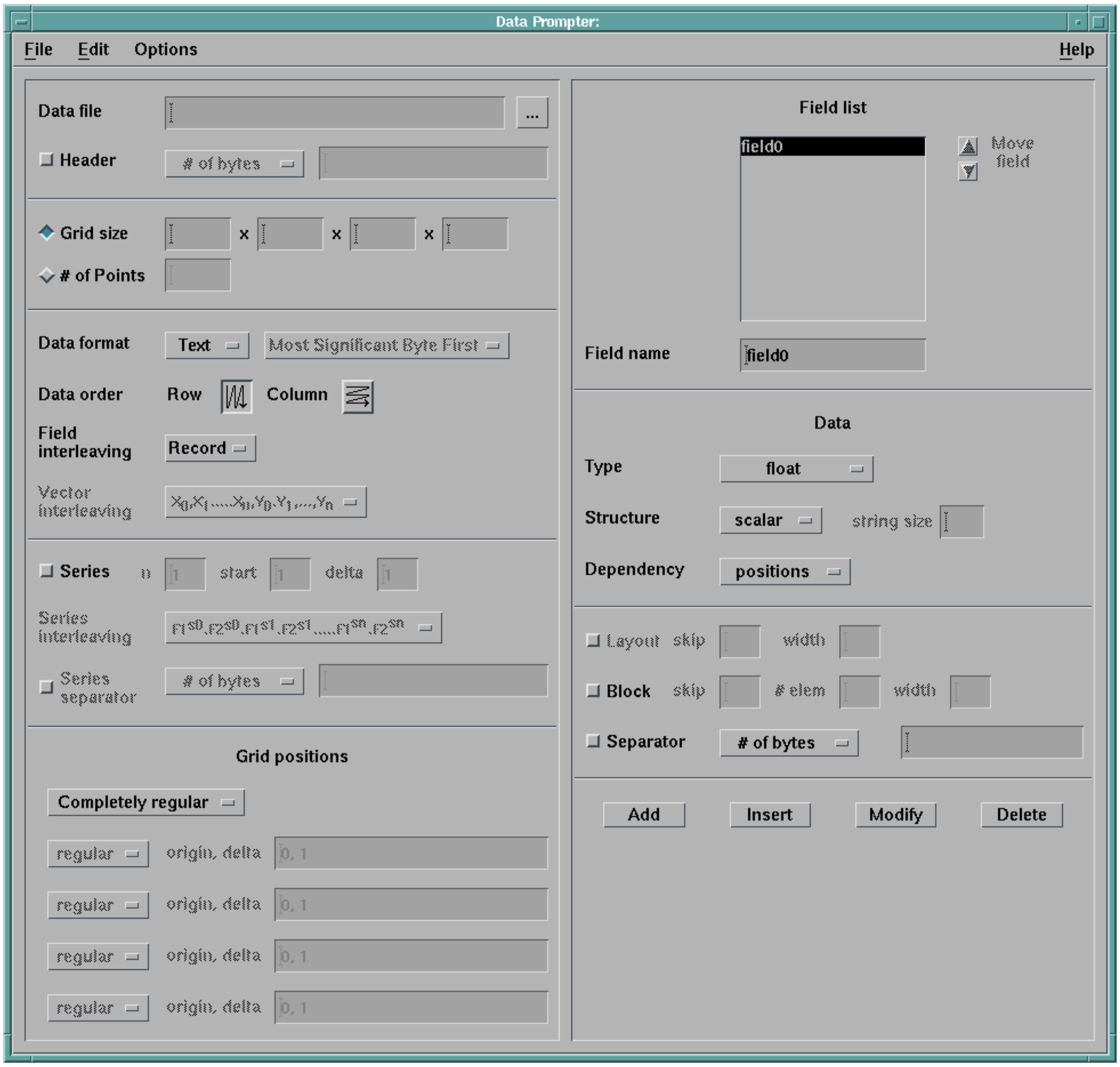
Data Order

This option specifies the layout of multidimensional arrays. **Row** (majority) means that the last index of a multidimensional array varies fastest (as in C language). **Column** (majority) means that the first index varies fastest (as in FORTRAN).

See also “majority” on page 92.

Field Interleaving

The data to be visualized must be organized in one of two general styles: block or columnar. For data laid out in blocks, select **Block** from the **Field Interleaving** option menu. For columnar style, select **Columnar**.



Data file

☐ Header # of bytes

◆ Grid size x x x

◇ # of Points

Data format Text Most Significant Byte First

Data order Row Column

Field interleaving Record

Vector interleaving $X_0,X_1,\ldots,X_n,Y_0,Y_1,\ldots,Y_n$

☐ Series n 1 start 1 delta 1

Series interleaving $F1^{s0},F2^{s0},F1^{s1},F2^{s1},\ldots,F1^{sn},F2^{sn}$

☐ Series separator # of bytes

Grid positions

Completely regular

regular origin, delta 0, 1

regular origin, delta 0, 1

regular origin, delta 0, 1

regular origin, delta 0, 1

Field list

field0

Field name field0

Data

Type float

Structure scalar string size

Dependency positions

☐ Layout skip width

☐ Block skip # elem width

☐ Separator # of bytes

Figure 17. Full Data Prompter. The two halves of this dialog box are described in separate sections in the text (see "Data File and Data Information" on page 103 and "Data Fields Information" on page 107).

Note that when **Columnar** is selected, both **Vector Interleaving** and **Series Interleaving** are grayed out. (See "Some Notes on General Array Importer Format" on page 66 for more information about the block and columnar styles.)

See also "interleaving" on page 90.

Vector Interleaving

The interpretation of vector data organized in records (blocks) is specified by the **Vector Interleaving** option button. The two choices in the associated option menu are:

$X_0Y_0, X_1Y_1, ..., X_nY_n$ "Vectors together"

and

$X_0, X_1, ..., X_n, Y_0, Y_1, ..., Y_n$ "Components together"

“Vectors together” means that all of the components of one vector are written together, then all the components of the next vector, and so on. “Components together” means that all of the x-components are written together, then all the y-components, and so on.

See also “interleaving” on page 90.

Series

For data consisting of a sequence of fields, click on the **`Series`** toggle button, activating the three text fields to the right:

- **`n`** specifies the number of series members in the data file.
- **`start`** specifies the series value for the first series member.
- **`delta`** specifies the difference in position between successive series members.

See also “series” on page 93.

Series Interleaving

Activating the **`Series`** toggle button also activates the **`Series Interleaving`** option menu if **`Field Interleaving`** is set to **`Record`**. The option menu contains two choices:
$\mathtt{F1}^{s0}, \mathtt{F2}^{s0}, \mathtt{F1}^{s1}, \mathtt{F2}^{s1}, \ldots, \mathtt{F1}^{sn}, \mathtt{F2}^{sn}$ “Series members together”
and
$\mathtt{F1}^{s0}, \mathtt{F1}^{s1}, \mathtt{F2}^{s0}, \mathtt{F2}^{s1}, \ldots, \mathtt{Fn}^{s0}, \mathtt{Fn}^{s1}$ “Fields together”

This feature is useful for series that consist of more than one field. “Series members together” means that the data for each field of series member zero (s0) are followed by all the data for each field for series member one (s1), and so on. “Fields together” means that all the series members for field 1 (F1) are followed by all the series members for field 2 (F2), and so on.

See also “interleaving” on page 90.

Series Separator

This feature is required only if the data for series members are separated from one another by non-data (e.g., comments). The specification is identical to that of the **`Header`** keyword (see Header on page 104).

See also “series” on page 93.

Grid Positions

This feature is required for gridded data if you have not specified that the positions are stored in the file (by naming one of the fields with the reserved word “locations”).

The entire section is grayed out if the “locations” reserved word is used. If the data consist of unconnected points (and are so specified in the initial dialog box or by the **`# of points`** toggle), all but one of the origin-delta fields is grayed out. See also “positions” on page 94.

The option button just to the right of the **`Grid positions`** (or **`Point positions`**) title offers the following choices:

`Completely regular`

The dimensions of the grid are all regular. This selection fixes the regular/irregular option button to the left of each dimension field at “regular” (i.e., “irregular” cannot be selected for any dimension). There will be as many origin/delta fields enabled as there are dimensions as specified by **`Grid size`**, or one if **`# of Points`** is chosen.

`Partially regular`

Each dimension of the grid can be either regular or irregular. This selection specifies a product of arrays. Thus, for example, the grid may be spaced equally in one dimension, but have an explicit list of positions in another dimension. For each dimension, you can set the **`regular`**/**`irregular`** option menu to the appropriate choice, and then enter either the origin, delta pair for that dimension or the explicit list of positions for that dimension.

Notes:

1. A regular dimension is specified completely by an origin and a delta (the number of positions in the dimension is specified in the grid section).
2. Specification of an irregular dimension requires an explicit and complete list of values (the number of items in the list must exactly equal the number of positions in the dimension).

`Explicit position list`

The positions are completely irregular and must be explicitly specified. If this option is chosen, then a single field is enabled for you to enter the position list. The number of items in the list should be equal to the total number of positions x the dimensionality of the positions.

Data Fields Information

Note: A change made in any option in this part of the dialog box generates an instruction to save (confirm) the change by clicking on the **`Modify`** button at the bottom of the panel.

Field List

This list (at the top right of the Data Prompter) displays the names of the fields that are currently defined for a header file.

If the list contains more than one field, their order must match that of the fields in the data file. To change the order of the fields, use the **`Move field`** arrows, after first selecting (highlighting) the field name to be moved.

Note that the settings of the various buttons and associated text fields below the field list are updated whenever a field is selected.

See also “field” on page 88.

Field name

The text field immediately below **`Field List`** displays the name of the current (selected) field. Field names must be unique. Default field names take the form “field*n*”, where *n* is an integer.

You can change a name and then click on the **`Modify`** button (near the bottom of the Data Prompter) to confirm the change. (Similarly you can add, insert, or delete a field. See **`Field List Buttons`** at the end of this section.)

Type

For each field in the list, select the appropriate data **`Type`** with the associated option button. The type must match that of the data in the field. The supported types are:

`double`	`byte`	`int`	`short`
`float`	`signed byte`	`signed int`	`signed short`
`string`	`unsigned byte`	`unsigned int`	`unsigned short`

Note that:

- byte is equivalent to unsigned byte
- int is equivalent to signed int
- short is equivalent to signed short.

Specifying `string` enables “string size,” which should contain the length of the longest string.

See also “type” on page 94.

Structure

For each field in the list, select the appropriate **`Structure`** with the associated option button. Accepted values are **`scalar`** and **`2-vector, ..., 9-vector`**. However, **`5-vector, ..., 9-vector`** cannot be specified for column-majority arrays.

See also “structure” on page 94.

Dependency

For each field in the list, select the appropriate **`Dependency`** with the associated option button. The default setting of this option is “positions” (one data item per position). If the data in the field are cell-centered (connection dependent) select “connections”.

See also “dependency” on page 88.

Layout

This option is automatically activated when the data in the current field are field-interleaved (columnar) ASCII (Text). It specifies the locations of data items in a line of text.

> **`skip`** specifies the number of characters the Importer must skip before it begins reading data in this field.
>
> **`width`** specifies the “width” (including blanks) of the field from which the data is to be read.

See also “layout” on page 92.

Block

This option is automatically enabled when the data in the current field are record-interleaved (block) ASCII (Text). It specifies the locations of data items in a line or field of text.

> **`skip`** specifies the number of characters the Importer must skip before it begins reading data in this field.
>
> **`# elem`** specifies the number of data elements to be read after a **`skip`**.
>
> **`width`** specifies the “width” (including blanks) of the data component.

See also “block” on page 88.

Field List Buttons

Four buttons at the bottom of the Data Prompter operate on the fields list:

Add Adds the field specified in **`Field name`** to the field list, placing it immediately *after* the current (highlighted) field. The settings for the new field are those current at the time of the addition.

Insert Inserts the field specified in **`Field name`** in the field list, placing it immediately *before* the current (highlighted) field. The settings for the new field are those current at the time of the insertion.

Modify Saves any change(s) to the settings of the current (selected) field.

Delete Removes the selected (highlighted) field from the field list.

Note: **Modify** and **Delete** are grayed out if there is no selected item in the field list.

Record Separator
This option is enabled only when **Block** field interleaving is selected. It allows you to specify a separator between blocks, or records, in the data file, when there are more than one. You can specify the same separator for between each record, or a different separator between each record. Depending on the setting of **Vector interleaving** button, separators may be specified between each field, or between each component of each vector in each field.

5.5 Data Prompter Browser

The Browser is a file-viewing tool to help you determine details of the layout and organization of a data file. Specifically, the Browser can measure offsets from a particular point (the top of a file, the beginning of a line, or a marker) and display the results in bytes and lines. These results can be copied and pasted (with standard Motif Copy/Paste operations) into the appropriate sections of the Data Prompter.

Starting the Browser

To browse a particular file:

1. Enter the path name of the file in the **Data file** field of either the simplified or the full Data Prompter.

   You can enter a name directly in the field or click on the ellipsis button (**...**) to the right and select a name from the **File Selection Dialog...** box.

2. Click on the ellipsis button again and select **Browser...**. The File Browser window appears, displaying the selected file (Figure 18 on page 110).

The File Browser window consists of three areas:

- The menu bar provides access to most of the Browser commands.
- The text window displays the contents of the selected data file.
- The offset area displays (in number of lines and bytes) the “distance” (or offset) between the cursor and a specified starting point.

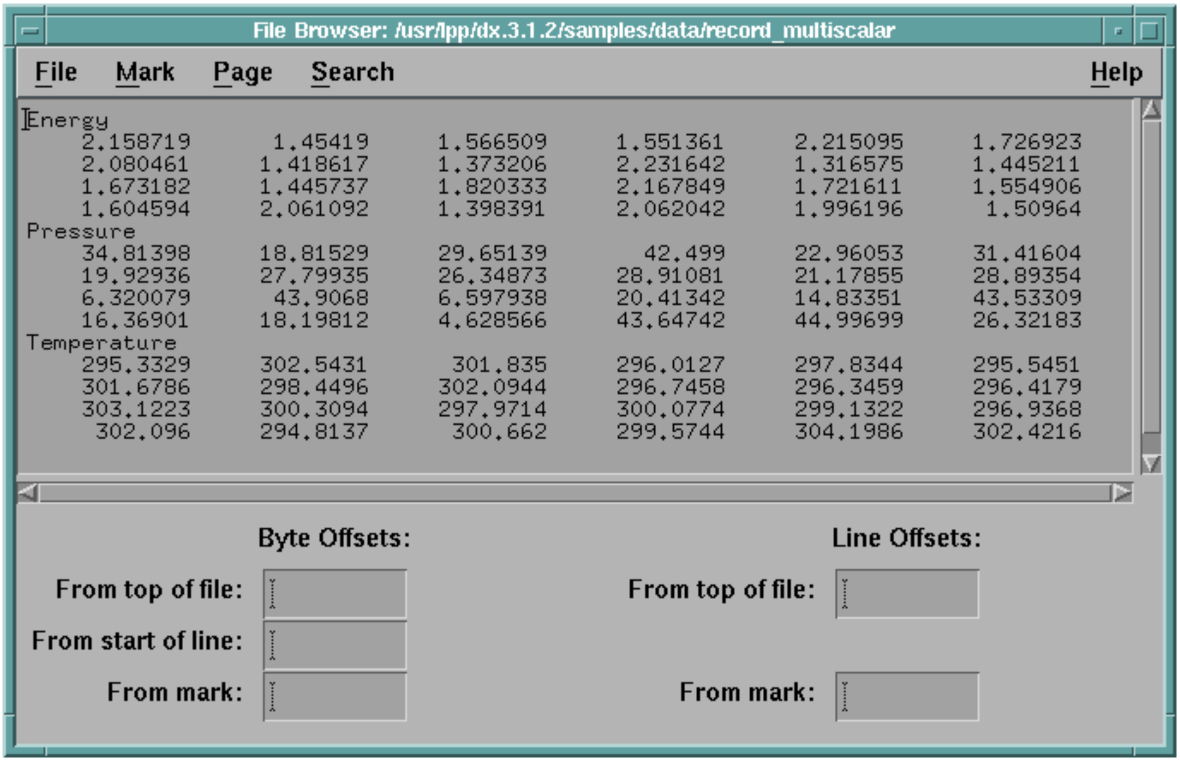
Browser Menu Bar

Browser File Menu

The single command in this menu is **Close**, which removes the Browser from the screen.

Browser Mark Menu

The commands in this menu control the placement of a “mark” in the selected file. The numbers displayed in the offset area are useful in specifying offset values for **header** (see Header on page 104), **layout** (see Layout on page 108), and **block** (see Block on page 108) in the Data Prompter.



```
Energy
     2.158719      1.45419     1.566509     1.551361     2.215095     1.726923
     2.080461     1.418617     1.373206     2.231642     1.316575     1.445211
     1.673182     1.445737     1.820333     2.167849     1.721611     1.554906
     1.604594     2.061092     1.398391     2.062042     1.996196      1.50964
Pressure
     34.81398     18.81529     29.65139       42.499     22.96053     31.41604
     19.92936     27.79935     26.34873     28.91081     21.17855     28.89354
     6.320079      43.9068     6.597938     20.41342     14.83351     43.53309
     16.36901     18.19812     4.628566     43.64742     44.99699     26.32183
Temperature
     295.3329     302.5431      301.835     296.0127     297.8344     295.5451
     301.6786     298.4496     302.0944     296.7458     296.3459     296.4179
     303.1223     300.3094     297.9714     300.0774     299.1322     296.9368
      302.096     294.8137      300.662     299.5744     304.1986     302.4216
```

Figure 18. File Browser Window. This window has just been opened and the cursor has not been moved from its initial position at the beginning of the file. No numbers will appear in the offset fields until the cursor is moved. The file displayed in the window is `/usr/lpp/dx/samples/data/record_multiscalar` *(see Example 1 of "Record Style: Multivariable Data" on page 75).*

Place mark
Replaces the character to the right of the cursor with a solid diamond-shaped mark. It will remain in place until moved to another position (with a new Place Mark command) or removed altogether. Note that the Browser allows only one mark at any time.

Clear mark
Removes a mark from the file.

Goto mark
Causes the Browser to display the text associated with a mark (if it is not already displayed in the file window).

Browser Page Menu

The two commands in this menu position the Browser at the beginning of the selected file (**`First page`**) or at the end of the file (**`Last page`**).

Browser Search Menu

The single command in this menu is **`Search...`**, which displays a search dialog box. Any string or regular expression can be entered in the **`Search for`** field. The search begins at the current position of the cursor and proceeds forward (the **`Find Next`** button) or back (the **`Find Previous`** button). Searches "wrap around" in either direction. The **`Close`** button closes the dialog box.

Browser Text Window

The File Browser window displays the contents of the file being browsed. The cursor can be positioned anywhere in the text (with the left mouse button). When the cursor is repositioned, the values in the offset area below the file window are updated.

Browsing Large Data Files

If the file in the text window exceeds a certain size, the Browser displays it in "chunks," the scroll bar controlling only the current chunk. When the slider is moved to the bottom of the scroll bar, the Browser reads in the next chunk; to top of the scroll bar, the preceding chunk.

Browser Offset Area

Five text fields at the bottom of the Browser display the "offsets" (in bytes and lines) from the top of the file or from a "mark" to the current position of the cursor. When the cursor is repositioned, the offset values are updated.

The text fields can be used to reposition the cursor anywhere in the file: enter a new value in a text field and press the Enter key. The cursor moves to the position specified by the new offset.

Copying Offset Values to the Data Prompter

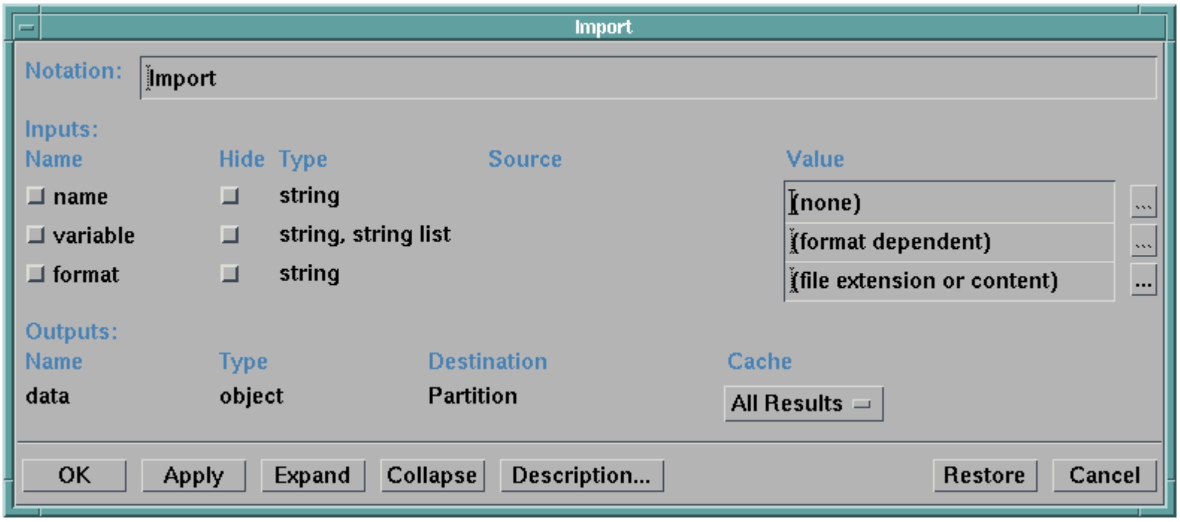
1. Highlight an offset value by triple clicking on the value field or dragging the cursor over the value.
2. Position the cursor in the target field in the Data Prompter and click once with the *middle* mouse button. The offset value appears in the target field.

5.6 Using the Header File to Import Data

Once you have created a Data Prompter header file, the next step is to use that file to import the data it describes. The basic procedure follows:

1. Place an Import tool icon on the VPE canvas (see "Selecting tools and placing icons" on page 22).
2. Double click on the icon to invoke the configuration dialog box (see Figure 19 on page 112).
3. The following information is required:
    a. The (path) **`name`** of the header file to be imported (in the first text field of the **`Value`** column (at right). If the Import module is part of a network displayed in the VPE, the path name appears automatically.
    b. Set **`format`** (the third field of the column)
    c. Set **`format`** (the third field of the column) to "general" unless the header file already has that as its extension.

Figure 19. Import Configuration Dialog Box.



Notation: Import

Inputs:

Name	Hide	Type	Source	Value
☐ name	☐	string		(none)
☐ variable	☐	string, string list		(format dependent)
☐ format	☐	string		(file extension or content)

Outputs:

Name	Type	Destination	Cache
data	object	Partition	All Results

Notes:

a. Header files created with the Data Prompter are given this extension automatically when they are saved with Save As... (Step 10 of Example 1 in To save the header file on page 68).

b. The **`variable`** parameter can be used to import a subset of the variables specified by the **`field`** keyword statement. For the General Array format, by default, all variables are imported.

The specified data file will now be imported when any visualization program containing this Import module is run.

Glossary

Some of the definitions in this glossary are taken from the *IBM Dictionary of Computing*, ZC20-1699.

A

accelerator. A keystroke sequence that reduces the number of steps needed to complete a task.

anchor window. The window in which a Data Explorer session starts (either the Visual Program Editor or the Image window). The window is identified by an anchor symbol in the top left corner. When this window is closed, the Data Explorer session ends.

array. In Data Explorer, an array structure containing an ordered list of data items of the same type along with additional descriptive information.

B

Browser. A Data Explorer user interface for determining the layout and organization of data in a file.

C

camera. An object that describes the viewing parameters of an image (e.g., width of the viewport, viewer's location relative to the object, and the resolution and aspect ratio of the image). A camera may be explicitly defined and passed as a parameter to the Render or Display module. It may also be implicitly defined in the use of interactive, mouse-driven options (such as zoom or rotate) in the Image window.

canvas. The area of a VPE window used in building and editing visual programs.

cell-centered data. See *connection-dependent data*.

clipping plane. A plane that divides a 3-dimensional object into a rendered and an unrendered region, making the object's interior visible.

colormap. In Data Explorer, a color map that relates colors to data values. The colors are carried in the map's "data" component and the data values to which each color applies in its "positions" component.

Colormap Editor. A Data Explorer user interface for mapping precise colors to specified data values, the results of which are displayed in a visual image.

component. A basic part of a field (such as "positions," "data," or "colors"); each component is indexed by a string (e.g., "positions"), and its value is typically an array object (e.g., the list of position values).

connection. Component of an IBM Data Explorer data field that specifies how a set of points are joined together. Also controls interpolation.

connection-dependent data. Cell-centered data. The data value is interpreted as constant throughout the connection element.

contour. On a surface, a line that connects points having the same data value (e.g., pressure, depth, temperature).

control panel. A Data Explorer interactive window that facilitates setting and changing the parameters of visual programs.

cube. A volumetric connection element that connects eight positions in a data field.

cutting plane. An arbitrary plane, in 3-dimensional space, onto which data are mapped.

D

data-driven interactor. Interactors whose attributes (such as minimum and maximum) are set by an input data field.

Data Prompter. A graphical user interface that enables a user to describe the format of the data in a file. The prompter creates a General Array Format header file that is used by the Import module to import the data.

dialog box. The "window" displayed when the user selects a pull-down option that offers or requires more detailed specification.

display. (1) *v.* To present information for viewing, usually on a terminal screen or a hard-copy device. (2) *n.* A device or medium on which information is presented, such as a terminal screen. (3) Deprecated term for *panel*.

Display module. A non-interactive module for displaying an Image. See Image window.

© Copyright IBM Corp. 1991-1997

E

element. See *connection*.

executive. The component of the Data Explorer system that manages the execution of specified modules. The term often refers to the entire server portion of the Data Explorer client-server model, including the executive, modules, and data-management components.

F

field. A self-contained collection of data items. A Data Explorer field typically consists of the data itself (the "data" component), a set of sample points (the "positions" component), a set of interpolation elements (the "connections" component), and other information as needed.

G

general array format. A data-importing method that uses a header file to describe the data format of a data file. This "format" makes it possible to import data in a variety of formats.

glyph. A graphical figure used to represent a particular variable. Each occurrence of a glyph represents a single value of the associated variable. Some attribute of the glyph (e.g., length or angle) is a function of the variable and varies with it.

group. A collection of objects.

I

icon. A displayed symbol that a user can point to with a device such as a mouse to select a particular operation or software application.

Image window. IBM Data Explorer window that displays the image generated by a visual program. Associated with the Image window are special interactors for 3-D viewing.

interactor. A Data Explorer device used to manipulate data in order to change the visual image produced by a program. See also *data-driven interactor, interactor stand-in*.

interactor stand-in. An icon used in the VPE window to represent an interactor. Stand-ins are named after the type of data they generate:

- integer
- scalar
- selector (outputs a value and a string)
- string
- value
- vector.

interpolation element. An item in the connections component array. Each interpolation element provides a means for interpolating data values at locations other than the specified set of sample points. See *positions component*.

invalid. A classification of an array item (typically positions or connections). An invalid item is not rendered or realized.

isosurface. A surface in 3-dimensional space that connects all the points in a data set that have the same value.

isovalue. The single value that characterizes each and every point constituting an isosurface. By default, this value is the average of all the data values in the set being visualized.

item. A single piece of data in an array.

L

line. An element that connects two positions in a field.

M

macro. In IBM Data Explorer, a sequence of modules that acts as a functional unit and is displayed as a single icon. Macros can also be defined in the Data Explorer scripting language.

menu bar. In windows, a horizontal bar that displays the names of one or more menus (or tasks). When the user selects a menu, a pull-down list of options for that menu is displayed.

module. (1) In IBM Data Explorer, a primitive function, such as Isosurface. (2) In a VPE window, the icon for a module (also called a *tool*, q.v.). (3) A program unit that is functionally discrete and identifiable (i.e., it can be assembled, compiled, combined with other units, and so on).

N

navigate. To move the camera (changing the "to" and "from" points) around the image scene, using the mouse.

netCDF. See *Network Common Data Form*.

network. In Data Explorer the set of tool modules, interactor stand-ins, and connections that constitute a visual program. In the VPE window, a network appears as a set of icons connected by arcs.

Network Common Data Form (netCDF). A data format that stores and retrieves scientific data in self-describing, multidimensional blocks (netCDF is not a database management system, however). netCDF is accessible with C and FORTRAN.

O

object. In IBM Data Explorer, any discrete and identifiable entity; specifically, a region of global memory that contains its own type-identification and other type-specific information.

opacity. The capacity of matter to prevent the transmission of light. For a surface, an opacity of 1 means that it is completely opaque; an opacity of 0, that it is completely transparent. For volume, opacity is defined as the amount of attenuation (of light) per unit distance.

P

palette. A displayed grouping of available selections (such as functions, modules, or colors) in a GUI window.

palindrome. In Data Explorer, a mode of running a sequence of images in one direction and then in the opposite direction.

pixel. Picture element. In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity.

position-dependent data. Data that are in one-to-one correspondence with positions.

positions component. A component that consists of a set of dimensional points in a field.

probe. A list of one or more vectors that represent points in a graphical image. Probes can be used with Data Explorer tools that accept vectors as input (such as ClipPlane and Streamline) or to control the view of an image.

pull-down. In windows, the list of options displayed when a task is selected from the menu bar.

Q

quad. An element that connects four positions in a field.

R

realization. A description of how raw data is to be represented in terms of boundaries, surfaces, transparency, color, and other graphical, image, and geometric characteristics.

rendering. The generation of an image from some representation of an object, such as a surface, or from volumetric information.

ribbon. A figure derived from lines (e.g., from streamlines and streaklines). Ribbons may twist to indicate vorticity.

S

sample point. A point that represents user data. Data is interpolated between sample points by interpolation elements (connections).

scalar. A non-vector value characterized by a single, real number.

scatter data. A collection of sample points without connections.

scripting language. The IBM Data Explorer command language. Used for writing visual programs, to manage the execution of modules, and to invoke visualization functions.

scroll. To move all or part of the display image vertically or horizontally to display data that cannot be observed in a single display image.

Sequencer. An IBM Data Explorer tool for creating “animated” sequences of images.

series. In IBM Data Explorer, used to represent a single field sampled across some parameter (e.g., a simulation of a CMOS device across a temperature range).

stand-in. See *interactor stand-in*.

streaklines. Lines that represent the path of particles in a changing vector field. Also called rakes.

streamlines. Lines that represent the path of particles in a vector field at a particular time. Also called flow lines.

T

tetrahedron. A volumetric connection element that connects four positions in a field.

tool. In IBM Data Explorer, a general term for any icon used to build a visual program (specifically, module, macro, or interactor stand-in).

triangle. A connection element that connects three positions in a field.

tube. A surface centered on a deriving line (e.g., a streamline or streakline). Tubes may twist to indicate vorticity.

V

vector. A quantity characterized by more than one component.

vertex. One of the positions that define a connection element.

visual program. A user-specified interconnected set of Data Explorer modules that performs a sequence of operations on data and typically produces an image as output.

Visual Program Editor (VPE). IBM Data Explorer window used to create and edit visual programs and macros. See also *canvas*.

volume. The amount of 3-dimensional space occupied by an object or substance (measured in cubic units). To be distinguished from an object's surface, which is a mathematical abstraction.

volume rendering. A technique for using color and opacity to visualize all the data in a 3-dimensional data set. The internal details visualized may be physical (such as the structure of a machine part) or they may represent characteristics (such as fluid flow, temperature, or stress).

vorticity. Mathematically defined as the curl of a velocity field. A particle in a velocity field with nonzero vorticity will rotate.

VPE. See *Visual Program Editor*.

Index

Numerics
2-D scalar glyphs 31
2-D streamlines 30
2-D vector glyphs 32
3-D scalar glyphs 34
3-D streamlines 33
3-D vector glyphs 34

A
accessing tutorials 5
adding captions 36
adding control points 17, 18
adding input tabs 37
animations, creating 41
arcs, connecting tool icons with 23
AutoAxes 13
AutoColor module 44
axes box 13

B
begin, note on where to 6
block keyword 80
block keyword statement 88
boxes, configuration dialog 38
Browser 1, 109
  and large data files 111
  copying offsets to the Data Prompter 111
  File menu 109
  Mark menu 109
  menu bar 109
  offset area 111
  Page menu 110
  Search menu 110
  starting 109
  text window, File 111
  window, File 109
browsing large data files 111

C
captions, adding 36
cell-centered data (example) 68, 76
Colormap Editor
  adding control points 17
  coloring data 29, 34, 44
  opening 16
  specifying values 17
  window 18
colors 29
column data order 104
column majority (data) format (example) 85
columnar style data format 81
Compute module 44
configuration dialog boxes 38
configuring AutoAxes 13
connecting scattered data points 37
connecting tool icons with arcs 23
context-sensitive Help 5
contour lines 29
control panels 2, 15
control panels, creating 39
control points 16, 17, 18, 20
controlling
  appearance of object 10
  execution, with Switch 37
  field of view 12
  inputs 38, 39
  rotation 11
  size of object 10
  view of object 10
  viewing direction 10
  visual image 10
creating animations 41
creating control panels 39
creating header files 66
creating macros 42

D
data (see also importing data)
  block keyword, using 80
  cell-centered 68, 76
  coloring 29
  coloring data 34, 44
  column majority 85
  deformed regular grid 78, 81
  dependency 64
  deriving grid information from 71
  describing 63
  examples, importing 67—85
  field, naming a 70
  fields 62
  format, General Array 66
  importing 25—29, 62—112
  model 1
  multiple scalar fields 75
  multiple scalars 76
  scalar and vector 77
  scalar and vector (regular grid) 81
  scalar, on a regular grid 67
  scattered 79

© Copyright IBM Corp. 1991-1997

data (see also importing data) *(continued)*
  scattered scalar 82
  series 73
  time series, with text 84
  vector 71
  visualizing 2-D 29
  visualizing 3-D 32
  warping 30
  with header in same file 74
  with header information 69
  with text 82
data and header in same file (example) 74
Data Browser (see Browser)
data examples (see importing data (examples))
Data Explorer data model 1
Data Explorer Data Prompter (see Data Prompter)
Data Explorer scripting language 1
data fields 62
data format
  record style 67, 75
data model, Data Explorer 1
data order 104
data points, connecting scattered 37
Data Prompter 1, 96
  File menu 109
  full 103
  initial dialog box 98
  Mark menu 109
  menus 101
  Page menu 110
  Search menu 110
  simplified 101
Data Prompter Browser (see Browser)
Data Prompter File Pull-down Menu 102
data visualization 1
data with text (example) 82
data-driven tools 43
deformed regular grid (example) 78, 81
deleting a tool icon 24
dependencies, mixed (example) 76
dependency keyword statement 88
dependency, data 64
deriving grid information (example) 71
describing data 63
dialog box
  add control points 19
  configuration 38
  Import configuration 111
  view control 11
Display module 2

E
editing a visual program 22
editor
  Colormap 16
editor *(continued)*
  Visual Program 2, 22
end keyword statement 96
examples, data (see importing data (examples))
executing a visual program 8

F
field interleaving 104
field keyword statement 88
field of view 12
field, naming (example) 70
fields, data 62
fields, multiple scalar (example) 75
File Browser text window 111
file keyword statement 86
File menu, Browser 109
format
  column majority (example) 85
  data (see data format)
  General Array data 66
  keyword statement 89
full Data Prompter 103

G
General Array data format 66
General Array Importer 63
glyphs
  2-D scalar 31
  2-D vector 32
  3-D scalar 34
  3-D vector 34
grid keyword statement 87
grid types 64
grid, deformed regular (example) 78

H
header and data in same file (example) 74
header file examples 67
header file syntax 85
header files, creating 66
header keyword statement 89
Help, context-sensitive 5

I
icon(s) (see tool icon(s))
Image window 2, 9, 10
images, printing 47
images, processing 46
images, saving 47
import configuration dialog box 111
Importer, General Array 63
importing data 25—29, 62—112
  configuration dialog box 111

importing data *(continued)*
  examples (see importing data (examples))
  header file used for 111
  in General Array format 25
importing data (examples)
  block keyword 80
  cell-centered 68, 76
  column majority 85
  columnar style 81
  deformed regular grid 78, 81
  multiple scalar fields 75
  multiple scalars (mixed dependencies) 76
  multivariable 75
  record style 67, 75
  scalar and vector 77
  scalar and vector (regular grid) 81
  scalar, on a regular grid 67
  scattered 79
  scattered scalar 82
  series 73
  single-variable 67
  time series with text 84
  vector 71
  warped (see deformed)
  with header in same file 74
  with header information 69
  with text 82
initial dialog box 98
input tabs, adding 37
interactors 39
interleaving 71
interleaving keyword statement 90
interleaving, field 104
isosurfaces 32

K

keyword statements
  block 88
  dependency 88
  end 96
  field 88
  file 86
  format 89
  grid 87
  header 89
  interleaving 90
  layout 92
  majority 92
  points 87
  positions 94
  recordseparator 92
  series 93
  structure 94
  type 94
keyword syntax 85

L

layout keyword statement 92
lines, contour 29

M

macros, creating 42
macros, sample 59
majority keyword statement 92
Map module 45
Mark menu, Browser 109
menus
  Browser 109
  Data Prompter 101
  File 109
  Mark 109
  Search 110
model, Data Explorer data 1
Module Builder 2
modules
  AutoColor 44
  Compute 44
  Display 2
  in Data Explorer 2
  in the VPE 7
  Map 45
  Plot 46
  RubberSheet 30
  Switch 37
moving a tool icon 24
multiple scalar fields (example) 75
multiple scalars (example) 76

N

naming a field (example) 70

O

object, controlling appearance of 10
offset area, Browser 111
opening a visual program 6
order, data 104
overview of Data Explorer 1

P

Page menu, Browser 110
Plot module 46
points keyword statement 87
positions keyword statement 94
printing images 47
processing images 46

Prompter, Data (see Data Prompter)

R

record interleaving 71
record-style data format 67, 75
record-vector interleaving 71
recordseparator keyword statement 92
regular grid, deformed (example) 78
rendering, volume 34
rotation, controlling 11
row data order 104
RubberSheet module 30

S

sample macros 59
sample visual programs 49
saving images 47
scalar and vector data (example) 77, 81
scalar data on a regular grid (example) 67
scalar data, scattered (example) 82
scalar fields, multiple (example) 75
scalar glyphs 34
scalars, multiple (example) 76
scattered data (example) 79
scattered data points, connecting 37
scattered scalar data (example) 82
scripting language, Data Explorer 1, 47
Search menu, Browser 110
separator, series 106
Sequencer 14
series data (example) 73
series interleaving 106
series keyword statement 93
series separator 106
simplified Data Prompter 101
single-variable data (example) 67
size, controlling object 10
slices 32
specifying colormap values 17
starting the Data Browser 109
starting tutorials 5
streamlines (2-D) 30
streamlines (3-D) 33
structure keyword statement 94
Switch module 37
syntax
  header file 85
  keyword 85

T

tabs, adding input 37
tasks 36
text, in times series 84
text, interspersed with data 82
time series with text 84
tool icon(s)
  connecting 23
  deleting 24
  moving 24
tools (see modules)
tools, data-driven 43
tools, tasks and 36
Tutorial I 4
Tutorial II 22
tutorials, accessing 5
type keyword statement 94
types of grid 64

U

using a header file 111
using Compute 44
using control panels 15
using Map 45
using Plot 46
using the block keyword (example) 80
using the Colormap Editor 16
using the Sequencer 14

V

vector and scalar data (example) 77, 81
vector data (example) 71
vector glyphs 32, 34
view control 10
view, field of 12
viewing direction, controlling 10
visual image, controlling 10
Visual Program Editor (VPE) 2, 22
  canvas 7
  palettes 7
  tool icons 7
  window 6
visual programs
  editing 22
  executing 8
  opening 6
  sample 49
visualization, data 1
visualizing 2-D data 29
visualizing 3-D data 32
volume rendering 34
VPE (see Visual Program Editor)

W

warped regular grid (example) 78, 81

window
  File Browser 110
  Image 9
  Visual Program Editor 6

Readers' Comments — We'd Like to Hear from You

IBM Visualization Data Explorer
QuickStart Guide
Version 3 Release 1 Modification 4

Publication No. SC34-3262-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name	Address
Company or Organization	
Phone No.	

Cut or Fold
Along Line

Readers' Comments — We'd Like to Hear from You
SC34-3262-02

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Thomas J. Watson Research Center/Hawthorne
Data Explorer Development
P.O. Box 704
YORKTOWN HEIGHTS, NY
USA 10598-0704

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

IBM®

Printed in U.S.A.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber

SC34-3262-02

Spine information:

IBM Data Explorer **QuickStart Guide** Version 3
Release 1 Modification 4 SC34-3262-02