The VisAD Java Class Library Developers Guide

4 November 1999

Table of Contents

1. Introduction
   - 1.1 System Availability
   - 1.2 Package Structure
   - 1.3 Authorship, Copyright, History and Support
2. Overview
   - 2.1 A Very Simple Application Example
   - 2.2 A Simple Application Example
   - 2.3 Flexible Design by Reduction to Elements
   - 2.4 The Value of Integrated Metadata
   - 2.5 Toolkit for Designing Interaction Techniques
3. Data Model
   - 3.1 MathTypes
     - 3.1.1 RealType Constructors
     - 3.1.2 TextType Constructor
     - 3.1.3 TupleType Constructor
     - 3.1.4 RealTupleType Constructors
     - 3.1.5 FunctionType Constructor
     - 3.1.6 SetType Constructor
     - 3.1.7 MathType Methods
     - 3.1.8 ScalarType Methods
     - 3.1.9 RealType Methods
     - 3.1.10 TupleType Methods
     - 3.1.11 RealTupleType Methods
     - 3.1.12 FunctionType Methods
     - 3.1.13 SetType Methods
     - 3.1.14 Application Example: Synthesizing MathTypes
     - 3.1.15 Application Example: Analyzing MathTypes
   - 3.2 Data Class Hierarchy
     - 3.2.1 Real Constructors
     - 3.2.2 Text Constructor
     - 3.2.3 Tuple Constructors
     - 3.2.4 RealTuple Constructors
     - 3.2.5 Field Constructors
     - 3.2.6 Data Methods
     - 3.2.7 Real Methods
     - 3.2.8 Text Methods
     - 3.2.9 Tuple Methods
     - 3.2.10 RealTuple Methods
     - 3.2.11 Function Methods

3.2.12 Field Methods
3.2.13 FieldImpl Method
3.2.14 Application Example: Synthesizing Fields
3.3 Units
3.3.1 Unit Methods
3.3.2 SI Variables
3.3.3 BaseUnit Methods
3.3.4 CommonUnit Variables
3.4 CoordinateSystems
3.4.1 CoordinateSystem Constructors
3.4.2 CoordinateSystem Methods
3.5 Sets
3.5.1 Defining Interpolation Algorithms by Extending the Set Class
3.5.2 The Delaunay Class for Irregular Sets
3.5.3 Set Constructors
3.5.3.1 DoubleSet and FloatSet Constructors
3.5.3.2 LinearSet Constructors
3.5.3.3 IntegerSet Constructors
3.5.3.4 GriddedSet Constructors
3.5.3.5 IrregularSet Constructors
3.5.3.6 ProductSet and UnionSet Constructors
3.5.4 Set Methods
3.5.5 SimpleSet Methods
3.5.6 Delaunay Constructors
3.6 ErrorEstimates
3.6.1 ErrorEstimate Constructors
3.7 AuditTrails
3.8 Missing Data
3.9 FlatFields - Data Operations and Efficiency
3.9.1 FlatField Constructors
3.9.2 FlatField Methods
3.10 Immutable Data
3.11 DataReferences
3.11.1 DataReference Constructors
3.11.2 DataReference Methods
3.12 Application Example: Arrays versus VisAD Functions
3.12.1 Subtracting Images as Pixel Arrays in C
3.12.2 Subtracting Images as Pixel Arrays in VisAD
3.12.3 Subtracting Images as Functions in VisAD
4. Visualization
4.1 ScalarMaps and DisplayRealTypes
4.1.1 Common Sense and ScalarMaps
4.1.2 DisplayRealType and DisplayTupleType Constructors
4.1.3 DisplayRealType Methods Useful for Extending DataRenderer
4.1.4 ScalarMap and ConstantMap Constructors

4.1.5 Generally Useful ScalarMap Methods
4.1.6 ScalarMap Methods Useful for Extending DataRenderer
4.1.7 ConstantMap Methods
4.1.8 ScalarMapListener Methods
4.1.9 ScalarMapEvent Methods
4.1.10 Application Example: ScalarMaps and ConstantMaps
4.2 DataRenderers and DisplayRenderers
4.2.1 Java3D DataRenderer and DisplayRenderer Constructors
4.2.2 Java2D DataRenderer and DisplayRenderer Constructors
4.2.3 DataRenderer Methods
4.2.4 DisplayRenderer Methods
4.2.5 DisplayRendererJ2D Method
4.3 Controls
4.3.1 Control Methods
4.3.2 ControlListener Methods
4.3.3 ControlEvent Methods
4.3.4 AnimationControl Methods
4.3.5 ColorControl Methods
4.3.6 ColorAlphaControl Methods
4.3.7 ContourControl Methods
4.3.8 FlowControl Methods
4.3.9 GraphicsModeControl Methods
4.3.10 ProjectionControl Methods
4.3.11 RangeControl Methods
4.3.12 ShapeControl Methods
4.3.12.1 The PlotText.render_label Method
4.3.13 ValueControl Methods
4.3.14 TextControl Methods
4.4 Mouse Interactions and Direct Manipulation
4.4.1 Changing Data Values by Re-drawing Data Depictions
4.4.2 Application Example: Interactive Scaling
4.5 ShadowTypes
4.6 The Display Class
4.6.1 Java3D Display Constructors
4.6.2 Java2D Display Constructors
4.6.3 Display Methods
4.6.4 DisplayImpl Methods
4.6.5 RemoteDisplayImpl Methods
4.6.6 DisplayListener Methods
4.6.7 DisplayEvent Methods
4.7 Shapes
4.7.1 VisADGeometryArray Shapes
4.7.2 The PlotText.render_label Method
5. Computational Cells
5.1 Cell Constructors

5.2 Cell Methods
5.3 ActionImpl Methods
6. Distributed Computing
6.1 Distributed Computing Guidelines and Cautions
6.2 Connecting to Remote Machines
6.2.1 RemoteServerImpl Constructors
6.2.2 RemoteServer Methods
6.2.3 RemoteServerImpl Methods
6.3 Application Example: Collaborative Direct Manipulation
7. File Format and Data Form Adapters
7.1 Extracting Metadata From Data Objects Returned by Data Form Adapters
7.2 General Design of Data Form Adapters
7.2.1 Form Methods
7.3 FITS Adapter
7.4 netCDF Adapter
7.5 HDF-EOS Adapter
7.6 GIF / JPEG Adapter
7.7 Vis5D Adapter
7.8 McIDAS Adapter
7.9 VisAD Adapter (serialized Java objects)
8. User Interfaces
8.1 VisAD User Interface Classes
8.1.1 VisADSlider Constructor
8.1.2 LabeledRGBWidget and LabeledRGBAWidget Constructors
8.1.3 LabeledRGBWidget and LabeledRGBAWidget Methods
8.1.4 SelectRangeWidget Constructor
8.1.5 AnimationWidget Constructor
8.1.6 ContourWidget Constructor
8.1.7 GMCWidget Constructor
9. Simplified Classes for Using VisAD
10. The VisAD Spread Sheet
10.1. Spread Sheet Classes
10.2. Features of the SpreadSheet User Interface
10.2.1 Basic Commands
10.2.2 Menu Commands
10.2.2.1 File Menu
10.2.2.2 Edit Menu
10.2.2.3 Setup Menu
10.2.2.4 Display Menu
10.2.2.5 Options Menu
10.2.3 Toolbars
10.2.3.1 Main Toolbar
10.2.3.2 Formula Toolbar
10.2.3.2.1 Description
10.2.3.2.2 How To Enter Formulas

10.3. Future Plans
11. Extending the VisAD Java Class Library
12. Application Examples
12.1 The DisplayTest Class
12.2 Visualizing the HSV Color CoordinateSystem
12.3 Collaborative GOES Satellite Sounding Analysis
12.4 A Steerable Shallow Fluid Model
12.5 A Simple Weather Simulation Visualizer
12.6 Image Animation Using Java2D
12.7 Earth Topography and Bathymetry
13. Caveats and Future Plans
13.1 JavaBean Components
14. For More Information and Help with Problems
15. References

Appendix A Constraints on ScalarMaps and MathTypes
Appendix B The GoesCollaboration Application Source Code

1. Introduction

This is the VisAD Java Class Library Developers Guide, describing the design and use of the VisAD Java class library for interactive analysis and visualization of numerical data. It also describes the design rationale, based on lessons learned from early mainframe visualization [5], interactive visualization [6], interactive computational steering [8], high-speed networks [7, 10], virtual reality [10], and supporting a broad user community [1, 9]. Key design decisions include:

1. The use of pure Java for platform independence and to support data sharing and real-time collaboration among geographically distributed users. Support for distributed computing is integrated at the lowest levels of the system.
2. A general mathematical data model that can be adapted to virtually any numerical data, that supports data sharing among different users, different data sources and different scientific disciplines, and that provides transparent access to data independent of storage format and location (i.e., memory, disk or remote).
3. A general display model that supports interactive 3-D, data fusion, multiple data views, direct manipulation, collaboration, and virtual reality.
4. Data analysis and computation integrated with visualization to support computational steering and other complex interaction modes.
5. Support for two distinct communities: developers who create domain-specific systems based on VisAD, and users of those domain-specific systems. VisAD is designed to support a wide variety of user interfaces, ranging from simple data browser applets to complex applications that allow groups of scientists to collaboratively develop data analysis algorithms.
6. Developer extensibility in as many ways as possible.

1.1 System Availability

The VisAD Java class library, including complete source code and installation instructions, is freely available from:

```
http://www.ssec.wisc.edu/~billh/visad.html
```

VisAD requires Java 1.2. VisAD displays are generated using either Java2D (included in Java 1.2) or Java3D. More information about these is available at:

```
http://java.sun.com/
```

1.2 Package Structure

The VisAD system consists of the following packages:

```
visad                       - the core VisAD package
visad.java3d                - Java3D displays for VisAD
visad.java2d                - Java2D displays for VisAD
visad.ss                    - the VisAD Spread Sheet
visad.util                  - VisAD user interface utilities
visad.data                  - VisAD data format adapters
visad.data.fits             - VisAD - FITS file adapter
visad.data.netcdf           - VisAD - netCDF file adapter
visad.data.netcdf.units     - units parser for netCDF adapter
visad.data.netcdf.in        - data input for netCDF adapter
visad.data.netcdf.out       - data output for netCDF adapter
visad.data.hdfeos           - VisAD - HDF-EOS file adapter
visad.data.hdfeos.hdfeosc   - native interface to HDF-EOS
visad.data.gif              - VisAD - GIF / JPEG file adapter
visad.data.vis5d            - VisAD - Vis5D file adapter
visad.data.mcidas           - VisAD - McIDAS file adapter
visad.data.visad            - VisAD (serial object) file adapter
```

The following packages are distributed with VisAD:

```
nom.tam.fits                - Java FITS file binding
nom.tam.util                - Java FITS file binding
nom.tam.test                - Java FITS file binding
ucar.netcdf                 - Java netCDF file binding
ucar.multiarray             - Java netCDF file binding
visad.paoloa                - GOES satellite analysis
visad.aune                  - shallow fluid model
visad.benjamin              - Milky Way galaxy model
```

The VisAD source distribution also includes a directory visad/examples that contains classes with the default package (i.e., no package statement).

VisAD is constantly being updated to fix bugs and add features and we don't even try to track all of these changes with VisAD version numbers. Rather, a file named 'DATE' is included in distribution jar files that gives the date and time the ditribution file was created. We will change VisAD version numbers as new features accumulate.

1.3 Authorship, Copyright, History and Support

VisAD was written by programmers at the University of Wisconsin Space Science and Engineering Center (SSEC) and at the Unidata Program Office. They are:

Bill Hibbard - SSEC (contact author: hibbard@facstaff.wisc.edu)
Steve Emmerson - Unidata
Curtis Rueden - SSEC
Tom Rink - SSEC
Dave Glowacki - SSEC
Tom Whittaker - SSEC
Tommy Jasmin - SSEC
Don Murray - Unidata
Nick Rasmussen - SSEC

The following people made substantial intellectual contributions to the design:

John Anderson - SSEC
Dave Fulker - Unidata

VisAD is freely available including source code. It is protected by copyright statements embedded in the source code and in the NOTICE and LICENSE files distributed with the source code.

The VisAD Java class library is actually VisAD version 2.0. VisAD versions 1.0 and 1.1 were written in C by Bill Hibbard, Brian Paul (of SSEC) and Andre Battaiola (while visiting SSEC from INPE/CPTEC in Brazil) [8, 9], with substantial intellectual contributions from Charles Dyer of the UW Computer Sciences Dept.

VisAD has adopted the UD Units library developed by Steve Emmerson of Unidata. [http://www.unidata.ucar.edu/packages/udunits/index.html].

VisAD borrows design ideas and code from the Vis5D system for interactive visualization of numerical simulations of weather and other environmental phenomena [6, 9, 10]. Vis5D was written in C by Bill Hibbard, Johan Kellum (of SSEC), Brian Paul, Andre Battaiola, Dave Santek (of SSEC) and Marie-Francoise Voidrot-Martinez (while visiting SSEC from METEO France).

Vis5D grew out of the 4-D McIDAS system [5, 6], which was part of Verner Suomi's McIDAS system for visualizing data from his weather satellites. The 4-D McIDAS was the 3-D (plus animation) analog of Tom Whittaker's 2-D graphics subsystem of McIDAS, which was the first interactive weather graphics system.

The development of this software has been supported by NASA, EPA, NSF (via Unidata and NCSA), NOAA, ARPA and DOE. We especially want to thank Joe Bredekamp of NASA, Cliff Jacobs of NSF and Larry Smarr of NCSA for their support of the Java VisAD.

2. Overview

This is an overview of how applications are constructed using VisAD. Throughout this guide, we will capitalize the proper names of VisAD classes such as Data and Display, in accordance with Java custom. A VisAD application is a network of:

1. Data objects: these may be simple real number values, text strings, vectors of real numbers, arrays such as images or grids, or complex hierarchies of data. They may include metadata for units, coordinate systems, complex sampling topologies, missing data indicators and error estimates, or they be simple values with minimal metadata. Data objects are described more thoroughly in Section 3. Section 3.12 explains the relation between data structures in VisAD and the C programming language.
2. Display objects: these generate interactive 3-D depictions of Data objects on a workstation screen or in immersive virtual reality (such as a CAVE, ImmersaDesk, or helmet). Display objects are linked to Data objects, so that Data depictions are updated whenever Data values change. Some Displays implement direct manipulation, which enables users to change Data values by re-drawing Data depictions. Displays on different machines may be linked to the same Data objects, in which case geographically distributed users may collaboratively visualize and manipulate the same Data. Displays are described more thoroughly in Section 4.
3. Cell objects: these are computations that are invoked whenever their input Data objects change value. They take their name from the cells of spread sheets. Like displays, Cells are linked to Data objects through DataReference objects (in fact, Displays and Cells both extend Action, the general class for objects whose actions are triggered by changing Data values). Cell objects are described more thoroughly in Section 5.
4. User interface (UI) objects: these are generally part of a UI component package such as AWT or JFC, although there are a few specialized utility UI components in the VisAD class library (described in Section 8). UI objects may also link to Data objects. Data values may be changed by UI events (for example, sliders may change the values of real number data objects), or UI components may link to Actions so that they update whenever Data object values change.
5. DataReference objects: these are pointers to Data objects. For example, in the statement "x = 3", x plays the role of a DataReference object and 3 plays the role of a Data object. The value of 3 cannot change just as many VisAD Data classes have values that cannot change (these are called immutable classes). So DataReference objects are necessary to represent variable data, just as the variable "x" is necessary in programming languages. Display, Cell and UI objects are linked to Data objects through DataReference objects. And DataReference objects would be used as symbol table entries in VisAD applications that implement programming language interpreters. DataReference objects are described more thoroughly in Section 3.11.

VisAD exploits Java Remote Method Invocation (RMI) so that Data, DataReference, Display, Cell and user interface objects may be linked together independent of their location on the network. Thus users at geographically remote workstations may collaborate by constructing Displays and linking them to the same Data object. Applications can be developed with VisAD that enable users to locate Data objects via web browsers and drag-and-drop them into Displays, link them into data analysis algorithms, and share visualizations of the results with colleagues at other locations. VisAD's use of RMI is described more thoroughly in Section 6.

The World Wide Web has created a shared network of generally passive text and image information. Distributed objects enabled by Java RMI will make this shared network much more active; that is, a network that includes execution threads. The VisAD system's general data model and thorough use of Java RMI provide a way to build a shared, active network of scientific data, displays and computations. This network could:

1. Change dynamically.
2. Have many simultaneous users with their own sets of display and user interface objects.
3. Have an indefinite life span, with users connecting and disconnecting but the basic network remaining.
4. Support numerous interacting execution threads.
5. Provide entrance points via web pages.

2.1 A Very Simple Application Example

We start with an application that reads a time sequence of images from a netCDF file and displays it with animation. There are only four executable lines of code in the application that have anything to do with VisAD: 1) creating the netCDF file reader, 2) reading the file, 3) creating a display of the file, and 4) linking the display into a JFrame. That's about as simple as a visualization program can get. Here's the full source code:

```
// import needed classes
import visad.*;
import visad.util.DataUtility;
import visad.java3d.DisplayImplJ3D;
import visad.data.netcdf.Plain;
import java.rmi.RemoteException;
import java.io.IOException;
import java.awt.*;
import javax.swing.*;

public class VerySimple {

  // type 'java VerySimple' to run this application
  public static void main(String args[])
         throws VisADException, RemoteException, IOException {

    // create a netCDF reader
    Plain plain = new Plain();

    // read an image sequence from a netCDF into a data object
    DataImpl image_sequence = plain.open("images.nc");

    // create a display for the image sequence
    DisplayImpl display = DataUtility.makeSimpleDisplay(image_sequence);

    // create JFrame (i.e., a window) for the display
    JFrame frame = new JFrame("VerySimple VisAD Application");

    // link the display to the JFrame
    frame.getContentPane().add(display.getComponent());

    // set the size of the JFrame and make it visible
```

```
    frame.setSize(400, 400);
    frame.setVisible(true);
  }
}
```

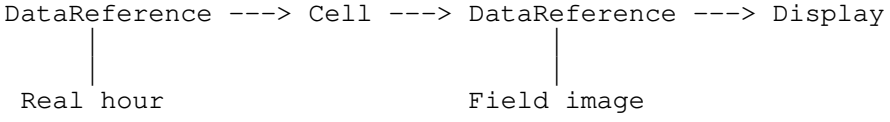
The VerySimple.java program is included in the visad/examples directory of the VisAD source distribution. To run it you also need to download and uncompress the images.nc file from:

```
ftp://iris.ssec.wisc.edu/pub/visad-2.0/images.nc.Z
```

into your visad/examples directory.

2.2 A Simple Application Example

The VerySimple application is so simple that it hides the network of VisAD objects it creates. Thus we present the Simple application which reads and displays the same image sequence, but provides some user interaction and makes the network of objects explicit. The diagram below shows the network of objects created by the Simple application. Its user controls a real number Data object (an hour value) via a UI slider, which in turn triggers a Cell to re-compute the value of a more complex Field Data object (for example, this may be an image array selected from an image sequence), whose depiction is updated in a Display.



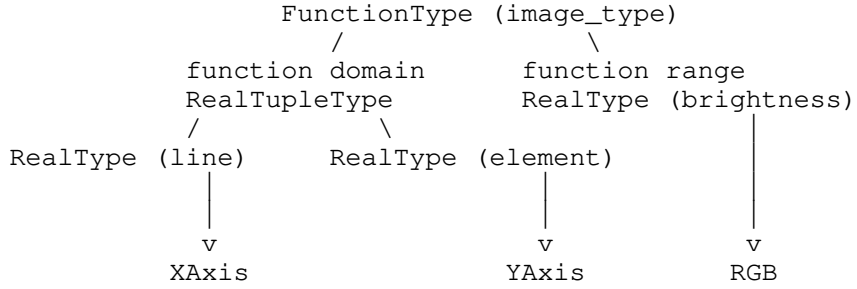
This diagram corresponds to the following simple application code:

```
// import needed classes
import visad.*;
import visad.java3d.DisplayImplJ3D;
import visad.util.VisADSlider;
import visad.data.netcdf.Plain;
import java.rmi.RemoteException;
import java.io.IOException;
import java.awt.*;
import java.awt.event.*;
import java.awt.swing.*;

public class Simple {

  // type 'java Simple' to run this application
  public static void main(String args[])
         throws VisADException, RemoteException, IOException {

    // create a DataReference for an image
    final DataReference image_ref = new DataReferenceImpl("image");

    // create a netCDF reader
    Plain plain = new Plain();
```

```
// open a netCDF file containing an image sequence and adapt
// it to a Field Data object
final Field image_sequence = (Field) plain.open("images.nc");

// create a Display using Java3D
DisplayImpl display = new DisplayImplJ3D("image display");

// extract the type of image and use
// it to determine how images are displayed
FunctionType image_sequence_type =
  (FunctionType) image_sequence.getType();
FunctionType image_type =
  (FunctionType) image_sequence_type.getRange();
RealTupleType domain_type = image_type.getDomain();
// map image coordinates to display coordinates
display.addMap(new ScalarMap((RealType) domain_type.getComponent(0),
                             Display.XAxis));
display.addMap(new ScalarMap((RealType) domain_type.getComponent(1),
                             Display.YAxis));
// map image brightness values to RGB (default is grey scale)
display.addMap(new ScalarMap((RealType) image_type.getRange(),
                             Display.RGB));

// link the Display to image_ref
// display will update whenever image changes
display.addReference(image_ref);

// create a DataReference and RealType for an 'hour' value
final DataReference hour_ref = new DataReferenceImpl("hour");
RealType hour_type =
  (RealType) image_sequence_type.getDomain().getComponent(0);
// and link it to a slider
VisADSlider slider = new VisADSlider("hour", 0, 3, 0, 1.0,
                                     hour_ref, hour_type);

// create a Cell to extract an image at 'hour'
// (this is an anonymous inner class extending CellImpl)
Cell cell = new CellImpl() {
  public void doAction() throws VisADException, RemoteException {
    // extract image from sequence by evaluating image_sequence
    // Field at 'hour' value
    image_ref.setData(image_sequence.evaluate(
                                   (Real) hour_ref.getData()));
  }
};
// link cell to hour_ref to trigger doAction whenever
// 'hour' value changes
cell.addReference(hour_ref);

// create JFrame (i.e., a window) for display and slider
JFrame frame = new JFrame("Simple VisAD Application");
frame.addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent e) {System.exit(0);}
});

// create JPanel in JFrame
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
panel.setAlignmentY(JPanel.TOP_ALIGNMENT);
panel.setAlignmentX(JPanel.LEFT_ALIGNMENT);
```

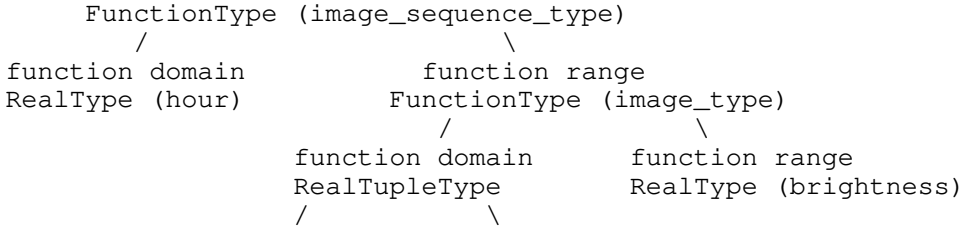
```
    frame.getContentPane().add(panel);

    // add slider and display to JPanel
    panel.add(slider);
    panel.add(display.getComponent());

    // set size of JFrame and make it visible
    frame.setSize(500, 600);
    frame.setVisible(true);
  }
}
```

Creating the DataReferences for 'hour' and 'image' and linking them to the VisADSlider and Cell is simple. Creating the Display and linking it to the 'image' DataReference is also simple. Setting up the JFrame and JPanel are not too difficult and really independent of VisAD. The only complex part of this application is extracting the image's type information for use in setting up the Display. Every VisAD Data object has a MathType that describes its basic structure. Every real number value occurring in a complex Data object has a RealType, a subclass of MathType, that includes a name like "latitude", "time" or "temperature". The code in our simple application extracts the RealTypes from the MathType of the image so that it can define different display roles for the real number values occurring in the image. The image Data object is interpreted as a function that maps pixel locations into pixel brightnesses, and its MathType, denoted image_type, is a FunctionType that includes MathTypes for the function's domain and range. The image_type can be diagrammed as:



Note that the bottom of the diagram includes the scalar mappings of image_type's RealType components to DisplayRealTypes: XAxis, YAxis and RGB (RGB indicates a pseudo color lookup table that maps brightness values to red, green and blue values).

The image_sequence Data object is treated as a function from time (hours) to images, so its MathType, denoted image_sequence_type, is also a FunctionType that can be diagrammed as:



```
RealType (line)     RealType (element)
```

Note that the image_type diagram is replicated in the range of this image_sequence_type diagram.

The call to the *getType* method of image_sequence returns its MathType, and then the calls to the *getRange*, *getDomain* and *getComponent* methods are used to parse the tree structure of the MathType to extract the RealTypes at the leaves of the tree. These RealTypes are then mapped to display coordinates such as Display.XAxis and Display.YAxis, and to display colors such as Display.RGB, using the ScalarMap constructors that are attached to the Display via its *addMap* method.

Note that image_sequence is treated as a function from a set of hour values to a set of images, and the *evaluate* method of image_sequence evaluates this function at an hour value and returns an image. Thus the *doAction* method of our computational Cell applies the *evaluate* method of image_sequence to an hour value to extract an image. Note also that image_sequence is declared as a Field, which is the VisAD class for functions represented by finite samplings.

In order to run the Simple application you need to download and uncompress the netCDF file "images.nc" from:

```
ftp://iris.ssec.wisc.edu/pub/visad-2.0/images.nc.Z
```

Response may be sluggish due to a problem with threads in early versions of Java3D. We should point out that the logic of this simple application, interactively selecting and displaying an image from an image sequence, can be implemented more simply and with faster response in a VisAD Display by mapping the "hour" RealType to Display.SelectValue. However, the Simple application is a nice illustration of how Data, DataReference, UI, Display and Cell objects can be linked together.

Section 12.3 describes a more complex application that creates a network of linked Data, DataReference, Display, Cell and UI objects distributed around the network to support collaboration among users at geographically remote locations. This application also includes direct manipulation Displays, where users change Data values by re-drawing their depictions. Appendix B is a complete source code listing of this application.

While the application described in Section 12.3 is more complex than the one presented here, it is still specific to a particular scientific problem. VisAD can be used to build much more flexible and generic applications. It would not be difficult to construct a generic spread sheet consisting of an array of Displays with one Data object per Display. UI components could let users add new Displays as needed and define the source of Data as: 1) a file, 2) direct manipulation in the Display, or 3) a mathematical expression involving Data objects in other Displays. VisAD could also be used as the basis for implementing a data flow system, or an interpreted numerical programming language.

2.3 Flexible Design of Applications by Reduction to Elements

The VisAD system offers a reductionist approach to design, as illustrated in the simple example of Section 2.2. Its image and image_sequence Data objects were defined as hierarchies of simple real values, and the Display for the image Data object was defined by mappings of its real values. This reductionist approach is very flexible in dealing with novel

applications. The VisAD data model, described in Section 3, enables developers to define many different numerical data structures in terms of hierarchies built up from simple real numbers and text strings, and enables developers to attach various types of metadata to values at different levels in the hierarchy.

The integration of metadata could allow a developer to define a sophisticated type for 2-D image data as finite samplings of continuous functions from 2-D pixel locations, such as (line, element) or (latitude, longitude), to one or more pixel radiances. Image metadata may include units for location and radiance values (e.g., radians or degrees for latitude and longitude locations), sampling topologies and geometries for pixel locations (most images have rectangular topologies, rectangular geometries in (line, element) locations but curvilinear geometries in (latitude, longitude) locations), coordinate systems for pixel locations (images with (line, element) locations may specify mathematical transformations to (latitude, longitude) locations), missing radiance indicators, and error estimates for pixel radiances and locations. Developers also have the option to ignore most of these types of metadata, and implement images as simple arrays without units, coordinate transformations, missing data or error estimates, and sampled on rectangular integer lattices (i.e., pixels are addressed by integer line and element indices, much as they are in Fortran or C arrays).

The VisAD display model offers a similar reductionist approach. Developers define displays for complex numerical data objects in terms of mappings (the ScalarMap class) from their primitive real number elements (the RealType class) to the conceptual elements of displays (the DisplayRealType class). Developers can also attach various types of display metadata and interactive controls to these mappings. Developers may even define new kinds of display elements by defining new DisplayRealTypes. This is described in detail in Section 4.

Designing VisAD data types and displays is similar to designing database schemas and views. In fact, most of the differences between VisAD data types and database schemas can be traced to the fact that databases model discrete entities while numerical data are discrete approximations to continuous entities.

VisAD's reduction to elements is very powerful for adapting to new applications, but, like database schema design, can also be a challenge. The power comes from providing a context in which developers can answer questions like "What is the nature of an image?" However, an end user who merely wants to display an image should not have to first answer such questions. Thus VisAD user interfaces should present choices to end users in higher-level terms such as images, grids and tables. Of course, it is possible to build user interfaces for VisAD that do defer such questions to end users, in order to give them the full power of the data model.

We also anticipate the development of intermediate class libraries between the core VisAD system and end user interfaces, which define higher-level application-specific data classes such as images, grids and tables. The methods of these higher-level data classes can encapsulate metadata manipulation in terms of higher-level data operations, including display methods that encapsulate manipulation of ScalarMaps from RealTypes to DisplayRealTypes. Such intermediate class libraries may simplify the task for those developing user interfaces for end users.

2.4 The Value of Integrated Metadata

The goal of integrating metadata is actually to create systems that enable end users to ignore metadata (but also to manipulate metadata if they wish to). For example, a user might read weather model output grids from several different models and several different file formats, each sampled at different map projections, at different vertical coordinate systems and at different time steps. The file format adapters will read each file into a VisAD Data object that includes the grid data and metadata objects containing the grid's spatial and temporal sampling information. Display objects will use these metadata objects to display the grid data co-located in space and time. Furthermore, arithmetical operations will also co-locate the data. For example, if temperatures from one model are subtracted from temperatures from another model, the temperatures from the second will be resampled to the spatial and temporal locations of the first before they are subtracted. If the two models use different temperature units, these will be converted before values are subtracted, and before they are displayed together.

Section 3.12 uses code examples to illustrate how VisAD can be used for simple array operations like those used in the C programming language, but can also be used for high-level operations on arrays of data that integrate metadata.

Users who want to control all aspects of their computations may do so by explicitly manipulating and extending the VisAD metadata classes. Note in particular Section 3.3 on Units, 3.4 on CoordinateSystems, and Section 3.5.1 on Defining Interpolation Algorithms by Extending the Set Class.

As the Internet enables greater data sharing among scientists, it increases the problems associated with metadata and file format differences among scientists. Metadata integration in a common data model is an important tool for addressing these problems, both for those users who want to ignore metadata and those who want to control metadata.

2.5 Toolkit for Designing Interaction Techniques

Interactivity is the key to understanding numerical data and computations. This has been the driving principal behind the development of Vis5D and VisAD. The most basic interaction mode is rotating 3-D scenes, which resolves the inherent ambiguity problem of 3-D graphics. That is, while 3-D graphics are more dramatic than 2-D graphics, they suffer from the problem that every point on a 2-D display screen or on the viewer's 2-D retinas corresponds to many points in the 3-D scene. Rotating the scene, whether in response to mouse movements for workstation displays or in response to head motion in immersive virtual reality displays, is the most effective way to resolve this ambiguity.

Once the necessary graphics speed is attained for interactive 3-D rotation, it can be exploited for all sorts of other interaction modes, such as dragging plane slices and other specialized graphics through data volumes, selecting various combinations of fields to visually compare, animating time dynamics, editing color maps, etc. VisAD supports all of these 'ordinary' graphical interaction modes when used with sufficiently fast graphics hardware.

When computations can also be done with fast response times, they may be coupled with interactive graphics to create an interaction mode known as 'computational steering'. By allowing Data, computational Cells, Displays and user interface components to be connected flexibly, VisAD supports computational steering interactions.

Beyond ordinary graphical interactions and computational steering, VisAD is designed to support a number of more sophisticated graphical interaction modes. These include:

1. Exploring visualization designs: experimenting with different ways to display the same data. VisAD allows users to determine how Data are depicted by defining a set of ScalarMaps from data primitives (i.e., RealTypes) to display primitives (i.e., DisplayRealTypes). Graphical user interfaces can be developed for defining ScalarMaps, enabling users to interactively experiment with display designs. For example, users might define ScalarMaps by dragging graphical icons representing RealTypes onto graphical icons representing DisplayRealTypes.
2. Direct manipulation: user interaction directly with data depictions. In particular VisAD allows users to modify Data values by re-drawing their depictions. While many ordinary graphical interactions have direct manipulation interfaces, they are usually not user-definable and have simple parameterizations in terms of one or a few real numbers. VisAD allows changes to larger Data objects to be connected through computational Cells and back to graphical Displays for more complex and user-defined graphical interactions.
3. Event driven computations and displays: re-computation and re-displays are triggered by data changes resulting from user interactions or running simulations. This extends the business spread sheet from simple numbers to complex numerical Data objects and their interactive 3-D visualizations. VisAD's Data, Display and Cell classes provide the tools for building numerical spread sheets.
4. Remote collaboration: geographically remote users share visualizations and user interfaces as if sitting in front of the same workstation. VisAD allows multiple remote Displays to share connections to a common set of Data objects and computational Cells.

Given this variety of basic interaction modes, VisAD can be viewed as a toolkit for building interaction techniques, in the same way that it and other systems are toolkits for building visualizations. The building blocks for interaction techniques are events, Display controls, direct manipulation, computational Cells, and shared access to Data across the network. Sections 4.4.2 and 6.3 present interesting small examples of building interaction techniques.

3. Data Model

The VisAD data model was designed to support virtually any numerical data. Rather than providing a variety of specific data structures like images, grids and tables, the VisAD data model defines a set of classes that can be used to build any hierarchical numerical data structures.

Data objects all have a class in the class hierarchy under Data, and all define a hierarchical composition of complex Data objects from primitive Data objects. The primitive (scalar) Data classes are Real and Text. A Real object contains a real number value (i.e., a member of **R**, the set of all real numbers) represented by a Java double. A Text object contains a text string. Complex hierarchical Data objects are built from these

primitives using the Tuple, Set and Function classes. A Tuple object contains a set of components whose number, sequence and type are fixed by the MathType of the Tuple. A Set object represents a set of points in an n-dimensional real vector space (denoted by **R**^n). There are a great variety of ways of representing such Sets, as described in Section 3.5. Note that a Tuple with n Real components is a RealTuple and represents a single point in **R**^n. A Function object represents a function from **R**^n to values of some specific type. Field is the subclass of Function for functions represented by finite sets of samples of function values (for example, a satellite image samples a continuous radiance function at a finite set of pixel locations). The Data classes implement methods for various binary and unary mathematical operations (e.g., add, multiply, sqrt), as well as specialized operations such as Function evaluation and Tuple component access. The Data class hierarchy is described in more detail in Section 3.2.

Data objects include metadata defined by the classes: MathType, Unit, CoordinateSystem, Set (function domain sampling), ErrorEstimate and AuditTrail, as well as missing data indicators. The details of these different forms of metadata are described in Sections 3.1 and 3.3 - 3.8. Metadata are integrated into mathematical and visualization operations. For example Unit conversions and CoordinateSystem transforms are done implicitly as needed in Data operations.

3.1 MathTypes

Numerical data objects are finite approximations to idealized mathematical objects such as real numbers, vectors, sets and functions. Thus every Data object has a MathType, which indicates the type of mathematical object that it approximates. The MathType class hierarchy is:

```
MathType
  ScalarType
    RealType
    TextType
  TupleType
    RealTupleType
  SetType
  FunctionType
```

The starting point for any new application of VisAD is defining a set of MathTypes for the Data objects involved. This set of MathTypes provides a context for defining metadata, data displays, and data analysis operations. This is similar to the way that database schemas provide a context for defining database applications. Developers using the VisAD class library can think about MathTypes using the following shorthand syntax:

```
MathType       := ScalarType | TupleType | SetType | FunctionType
ScalarType     := RealType | TextType
RealType       := name
TextType       := name
TupleType      := ( MathType , MathType , ..., MathType )
TupleType      := RealTupleType
RealTupleType  := ( RealType , RealType , ..., RealType )
SetType        := set ( RealTupleType )
FunctionType   := ( RealTupleType -> MathType )
```

```
FunctionType   := ( RealType -> MathType )
```

where *TupleType* and *RealTupleType* each have at least one component. For example, a satellite image of Earth may be a finite sampling of a continuous function with *MathType*:

```
( (latitude, longitude) ->
  (radiance_channel_1, ..., radiance_channel_N) )
```

The output of a weather model may be described using the *MathType*:

```
( time -> ( (latitude, longitude, altitude) ->
  (temperature, pressure, dewpoint, wind_u, wind_v, wind_w) ) )
```

And a set of map boundaries may be described using the *MathType*:

```
set ( (latitude, longitude) )
```

Note that the *prettyString* method of MathType returns a String with this shorthand notation for any VisAD MathType. The static *stringToType* method of MathType takes a String argument, which is assumed to be in this shorthand notation, and returns the corresponding MathType (of course, MathTypes returned by *stringToType* do not include any non-null default Units, CoordinateSystems or Sets).

MathTypes are a form of metadata that describe data organization. For example, weather model output are often stored in files as independent 2-D grids, where any higher-level organization must be deduced by comparing the metadata associated with each grid. MathTypes provide a way to explicitly document such higher-level data organizations.

Every scalar (i.e., primitive) value occurring in a Data object is associated with a named ScalarType occurring in the Data object's MathType. These names are used to control how the Data object is displayed, as described in Section 4.1.

Some MathTypes include default values for various kinds of metadata, including Units (see Section 3.3), CoordinateSystems (see Section 3.4), and samplings (see Section 3.5). Although these defaults may be over-ridden for Data values, the defaults define equivalence classes of convertible Units and CoordinateSystems among Data values with the same MathTypes, with convertibility enforced by the system. Note that application developers may opt out of Units, CoordinateSystems and any other form of metadata by setting that form of metadata to *null* in MathType and Data object constructors (however, developers may not opt out of MathTypes and Field samplings, which are mandatory).

MathType is abstract and serializable. A MathType object can only be local (see Section 6 for more information). Its subclasses are all immutable.

3.1.1 RealType Constructors

RealType includes the following constructors:

```
/** name of type (two RealTypes are equal if their names are equal);
    default Unit for values of this type and may be null; default Set
    used when this type is a FunctionType domain and may be null */
public RealType(String name, Unit default_unit, Set default_set)
       throws VisADException;
```

```
/** name of type (two RealTypes are equal if their names are equal);
    default Unit and Set are null */
public RealType(String name) throws VisADException;
```

3.1.2 TextType Constructor

TextType includes the following constructor:

```
/** name of type (two TextTypes are equal if their names are equal) */
public TextType(String name) throws VisADException;
```

3.1.3 TupleType Constructor

TupleType includes the following constructor:

```
/** array of component types */
public TupleType(MathType[] types) throws VisADException;
```

3.1.4 RealTupleType Constructors

RealTupleType includes the following constructors:

```
/** array of component types;
    default CoordinateSystem for values of this type (including
    Function domains) and may be null; default Set used when this
    type is a FunctionType domain and may be null */
public RealTupleType(RealType[] types,
                     CoordinateSystem default_coordinate_system,
                     Set default_set) throws VisADException;

/** construct a RealTupleType with one component */
public RealTupleType(RealType a,
                     CoordinateSystem default_coordinate_system,
                     Set default_set) throws VisADException;

/** construct a RealTupleType with two components */
public RealTupleType(RealType a, RealType b,
                     CoordinateSystem default_coordinate_system,
                     Set default_set) throws VisADException;

/** construct a RealTupleType with three components */
public RealTupleType(RealType a, RealType b, RealType c,
                     CoordinateSystem default_coordinate_system,
                     Set default_set) throws VisADException;

/** construct a RealTupleType with four components */
public RealTupleType(RealType a, RealType b, RealType c, RealType d,
                     CoordinateSystem default_coordinate_system,
                     Set default_set) throws VisADException;

/** array of component types;
    default CoordinateSystem and Set are null */
public RealTupleType(RealType[] types) throws VisADException;

/** construct a RealTupleType with one component */
public RealTupleType(RealType a) throws VisADException;

/** construct a RealTupleType with two components */
```

```
public RealTupleType(RealType a, RealType b) throws VisADException;

/** construct a RealTupleType with three components */
public RealTupleType(RealType a, RealType b, RealType c)
       throws VisADException;

/** construct a RealTupleType with four components */
public RealTupleType(RealType a, RealType b, RealType c, RealType d)
       throws VisADException;
```

3.1.5 FunctionType Constructor

FunctionType includes the following constructor:

```
/** domain must be a RealType or a RealTupleType;
    range may be any MathType */
public FunctionType(MathType domain, MathType range)
       throws VisADException;
```

3.1.6 SetType Constructor

SetType includes the following constructor:

```
/** domain must be a RealType or a RealTupleType */
public SetType(MathType domain) throws VisADException;
```

3.1.7 MathType Methods

Generally useful MathType methods include:

```
/** returns a missing Data object for any MathType */
public Data missingData() throws VisADException;

/** return a String that indents complex MathTypes
    for human readability */
public String prettyString();

/** return an array of ScalarMaps that is an "intuitive"
    guess at a good way to visualize this MathType;
    returns null if it can't make a good guess */
public ScalarMap[] guessMaps(boolean threeD);

/** ScalarTypes are equal if they have the same name;
    TupleTypes are equal if their components are equal;
    FunctionTypes are equal if their domains and ranges
    are equal */
public boolean equals(Object obj);

/** this is useful for determining compatibility of
    Data objects for binary mathematical operations;
    any RealTypes are equal; any TextTypes are equal;
    TupleTypes are equal if their components are equal;
    FunctionTypes are equal if their domains and ranges
    are equal */
public boolean equalsExceptName(MathType type);

/** create a MathType from its string represnetation;
    essentially the inverse of the prettyString method */
public static MathType stringToType(String s) throws VisADException;
```

3.1.8 ScalarType Methods

Generally useful ScalarType methods include:

```
public String getName();
```

3.1.9 RealType Methods

Generally useful RealType methods include:

```
/** return any RealType constructed in this JVM with name,
    or null */
public static RealType getRealTypeByName(String name);

/** get default Unit */
public Unit getDefaultUnit();

/** get default Set*/
public Set getDefaultSet();

/** this is a violation of MathType immutability to allow a
    a RealType to be an argument (directly or through a
    SetType) to the constructor of its default Set;
    this method throws an Exception if getDefaultSet has
    previously been invoked */
public void setDefaultSet(Set set) throws VisADException;
```

3.1.10 TupleType Methods

Generally useful TupleType methods include:

```
/** return number of components */
public int getDimension();

/** return component for index between 0 and getDimension() - 1 */
public MathType getComponent(int index) throws VisADException;

/** return index of first component with type;
    if no such component, return -1 */
public RealType getIndex(MathType) throws VisADException;

/** return index of first RealType component with name;
    if no such component, return -1 */
public RealType getIndex(String name) throws VisADException;
```

3.1.11 RealTupleType Methods

Generally useful RealTupleType methods include:

```
/** get default Units of RealType components */
public Unit[] getDefaultUnits();

/** get default CoordinateSystem */
public CoordinateSystem getCoordinateSystem()

/** get default Set*/
public Set getDefaultSet();
```

```
  /** this is an unavoidable violation of MathType immutability -
      a RealTupleType must be an argument (directly or through a
      SetType) to the constructor of its default Set;
      this method throws an Exception if getDefaultSet has
      previously been invoked */
  public void setDefaultSet(Set set) throws VisADException;
```

3.1.12 FunctionType Methods

Generally useful FunctionType methods include:

```
  /** if the domain passed to constructor was a RealType,
      getDomain returns a RealTupleType with that RealType
      as its single component */
  public RealTupleType getDomain();

  public MathType getRange();
```

3.1.13 SetType Methods

Generally useful SetType methods include:

```
  /** if the domain passed to constructor was a RealType,
      getDomain returns a RealTupleType with that RealType
      as its single component */
  public RealTupleType getDomain();
```

3.1.14 Application Example: Synthesizing MathTypes

Applications that construct Data objects from numerical values they compute generally need to synthesize MathTypes from their RealType components. Here's a sample of code for synthesizing a MathType appropriate for a Vis5D data set (this is roughly the inverse of the code in Section 3.1.15):

```
  // construct RealType components for grid coordinates
  RealType row = new RealType("row", null, null);
  RealType column = new RealType("column", null, null);
  RealType level = new RealType("level", null, null);

  // construct RealTupleType for grid coordinates
  RealType[] types3d = {row, column, level};
  RealTupleType domain = new RealTupleType(types3d);

  // construct RealType components for grid fields
  RealType temperature = new RealType("temperature", null, null);
  RealType pressure = new RealType("pressure", null, null);
  RealType water_vapor = new RealType("water_vapor", null, null);

  // construct RealTupleType for grid fields
  RealType[] field3d = {temperature, pressure, water_vapor};
  RealTupleType range = new RealTupleType(field3d);

  // construct FunctionType for grid
  FunctionType grid_type = new FunctionType(domain, range);

  // construct RealType and RealTupleType for time domain
  RealType time = new RealType("time", null, null);
  RealTupleType time_type = new RealTupleType(time);
```

```
// construct FunctionType for time sequence of grids
FunctionType vis5d_type = new FunctionType(time_type, grid_type);
```

3.1.15 Application Example: Analyzing MathTypes

Applications that get Data objects from file format adapters (described in Section 7) generally need to analyze MathTypes to extract their RealType components. The Vis5DForm class adapts Data objects from Vis5D files, whose MathTypes have the general form:

```
(time -> ((row, column, level) -> (field1, field2, ..., fieldN)))
```

That is, they are time sequences of multivariate 3-D grids. Here's a sample of MathType analysis code (this is roughly the inverse of the code in Section 3.1.14):

```
// get the MathType of a Data object named 'vis5d'
FunctionType vis5d_type = (FunctionType) vis5d.getType();

// extract time, the domain of the FunctionType
RealType time = (RealType) vis5d_type.getDomain().getComponent(0);

// get grid_type, itself a FunctionType and the range of the
// vis5d_type FunctionType
FunctionType grid_type = (FunctionType) vis5d_type.getRange();

// get the grid domain RealTupleType
RealTupleType domain = grid_type.getDomain();

// get the grid domain component RealType - they are grid coordinates
RealType row = (RealType) domain.getComponent(0);
RealType column = (RealType) domain.getComponent(1);
RealType level = (RealType) domain.getComponent(2);

// get the grid range - it is a RealTupleType of fields
RealTupleType range = (RealTupleType) grid_type.getRange();

// get the number of grid range components
int dim = range.getDimension();

// construct an array to hold the grid range RealTypes
RealType[] range_types = new RealType[dim];

// get the grid range RealTypes
for (int i=0; i<dim; i++) {
  range_types[i] = (RealType) range.getComponent(i);
}
```

3.2 Data Class Hierarchy

The Data hierarchy is:

```
Data
  Scalar
    Real
    Text
  Tuple
```

```
  RealTuple
Set
  (there is a large hierarchy under Set as described in Section 3.5)
Function
  Field
    FlatField
```

To some extent the Data hierarchy mirrors the MathType hierarchy. However, it is important to note that MathType is not a synonym for Data class, since Data classes may be elaborated into different forms of finite representation of the corresponding MathTypes. For example, Set is elaborated into a large number of different ways of representing subsets of **R^**n. Similarly, Function is elaborated into Field, for functions represented by finite samplings, and FlatField, for Fields with simple range values that can be represented by small numbers of Java's primitive data types rather than by objects. Developers may extend the Data classes to define new forms of representation. For example, a developer could extend Real to define a representation by ratios of infinite-precision integers rather than the Java primitive *double* used by Real (doubles are used by Real because experience has shown that using *floats* as the default can cause round-off problems that are extremely difficult for application developers to detect and diagnose).

The Data hierarchy is also elaborated for various data storage locations and formats. Section 6 describes how the hierarchy for Data and other VisAD classes is structured for local and remote objects, and Section 7 describes how the Data class hierarchy is adapted to import data from various file formats. The Data hierarchy is being adapted to netCDF, HDF and FITS files, and developers may extend this to other file formats. Thus data are accessible via the VisAD Data API (Application Programming Interface) independent of storage location, file format and approximating representation.

The metadata classes described in Sections 3.1 and 3.3 - 3.8 define how Data objects approximate mathematical objects and how they model the world.

Data is an interface that may apply to both local and remote Data objects. DataImpl is an abstract class that only applies to local Data objects, and RemoteData is an interface that only applies to remote Data objects (see Section 6 for more information). DataImpl is cloneable and serializable. All of its subclasses except FieldImpl and FlatField are immutable. API documentation for the Set class hierarchy is described in Section 3.5 and for FlatFields is described in Section 3.9, rather than here.

3.2.1 Real Constructors

Real includes the following constructors:

```
/** unit and error may be null */
public Real(RealType type, double value, Unit unit,
            ErrorEstimate error) throws VisADException;

/** use RealType.Generic */
public Real(double value)
```

3.2.2 Text Constructor

Text includes the following constructor:

```
public Text(TextType type, String value) throws VisADException;
```

```
/** use TextType.Generic */
public Text(String value)
```

3.2.3 Tuple Constructors

Tuple includes the following constructors:

```
/** this constructs its MathType from the MathTypes of the
    data array; components are copies of data */
public Tuple(Data[] data) throws VisADException, RemoteException;

/** only copy data if copy == true */
public Tuple(Data[] data, boolean copy)
       throws VisADException, RemoteException;
```

3.2.4 RealTuple Constructors

RealTuple includes the following constructors:

```
/** coordinate_system may be null; otherwise
    coordinate_system.getReference() must equal
    type.getCoordinateSystem.getReference() */
public RealTuple(RealTupleType type, Real[] reals,
                 CoordinateSystem coordinate_system)
       throws VisADException, RemoteException;

public RealTuple(Real[] reals) throws VisADException, RemoteException;
```

3.2.5 Field Constructors

Field is an interface implemented by FieldImpl for local Fields and RemoteFieldImpl for remote Fields. See Section 6 for more information about distributed computing. These classes have the following constructors:

```
/** FieldImpl is the most general sampled function;
    domain_set defines the domain sampling;
    if it is null, use the default Set of type.getDomain();
    domain_set defines the Units and CoordinateSystem
    of the Field domain */
public FieldImpl(FunctionType type, Set domain_set)
       throws VisADException;

/** use the default Set of type.getDomain() */
public FieldImpl(FunctionType type) throws VisADException;

/** construct a RemoteFieldImpl object to provide remote
    access to field */
public RemoteFieldImpl(FieldImpl field)
       throws VisADException, RemoteException;
```

3.2.6 Data Methods

A Data object may be either local or remote, a DataImpl object may only be local and a RemoteData object may only be remote (see Section 6 for more information). The methods in this section define the universal operations applicable to all Data objects: *getType* returns a Data object's MathType, *isMissing* indicates whether the Data object has missing value (but note that even is a Data object is not missing, it may still have sub-

objects with missing values), and *local* replaces a RemoteData object with a local DataImpl copy.

The *binary* and *unary* methods define basic mathematical operations on Data that are the building blocks for data analysis using VisAD. The *binary* and *unary* methods have wrapper methods for specific operations like *add* and *sin*. These operations are defined point-by-point for Tuple and Function Data objects, so that for example, the *sin* of a Function is a Function whose values are the sines of the original Function's values.

When *add* (or any other *binary* operation) is applied to two Fields the result is a Field whose values are the sums (or other operation) of the values of the two Functions, but only if the MathTypes of the two Fields match. MathType matching is defined recursively on TupleTypes and FunctionTypes in terms of their components, any RealType matches any RealType, and any TextType matches any TextType (thus matching Functions must have domains with the same dimension).

Most important, binary and unary operations on Data objects involve their metadata. When two Fields are added, the domain samples of one are resampled to the domain samples of the other, including any necessary Unit conversions of Real components of the domains and any necessary CoordinateSystem transformations between RealTuple domains. The range values of one Field are estimated at the domain sample locations of the other Field using either nearest neighbor or weighted average algorithms, as specified in the optional resampling_mode argument to *binary* methods. Unit conversions and CoordinateSystem transformations are also applied as needed to range values of Fields before they are added. Furthermore, ErrorEstimates attached to Field range values are modified to reflect error effects of *binary* and *unary* operations. ErrorEstimate propagation may assume either that operand errors are independently or dependently distributed, or ErrorEstimate propagation may be disabled, using the error_mode argument to *binary* and *unary* methods.

In some cases Data objects may be combined in *binary* operations even if their MathTypes do not match. For example, a Real object may be combined with any other Data object, and a Functions may be combined with Data objects that match the MathType of the Function's range.

```
public MathType getType()
       throws VisADException, RemoteException;

/** flag indicating whether Data object has missing value */
public boolean isMissing()
       throws VisADException, RemoteException;

/** if remote, return a local copy;
    if local, return this */
public DataImpl local()
       throws VisADException, RemoteException;

/** general binary operation between this and data; operation may
    be Data.ADD, Data.SUBTRACT, etc; these include all binary
    operations defined for Java primitive data types; new_type
    is the MathType of the result; sampling_mode may be
    Data.NEAREST_NEIGHBOR or Data.WEIGHTED_AVERAGE; error_mode
    may be Data.INDEPENDENT, Data.DEPENDENT or Data.NO_ERRORS */
public Data binary(Data data, int operation, MathType new_type,
```

```
                    int sampling_mode, int error_mode)
       throws VisADException, RemoteException;

/** like previous signature of binary, except the result takes
    the MathType of this unless the default Units of that MathType
    conflict with Units of the result, in which case a generic
    MathType with appropriate Units is constructed */
public Data binary(Data data, int operation, int sampling_mode,
                    int error_mode)
       throws VisADException, RemoteException;

public Data add(Data data, int sampling_mode, int error_mode)
       throws VisADException, RemoteException;

/** use Data.NEAREST_NEIGHBOR and Data.NO_ERRORS */
public Data add(Data data) throws VisADException, RemoteException;

public Data subtract(Data data, int sampling_mode, int error_mode)
       throws VisADException, RemoteException;

/** use Data.NEAREST_NEIGHBOR and Data.NO_ERRORS */
public Data subtract(Data data) throws VisADException, RemoteException;

/** similar methods are defined for the following binary operators:
    multiply, divide, pow, max, min, atan2, atan2Degrees and
    remainder */

/** general unary operation; operation may be Data.ABS, Data.ACOS, etc;
    these include all unary operations defined for Java primitive data
    types; new_type is the MathType of the result; sampling_mode may be
    Data.NEAREST_NEIGHBOR or Data.WEIGHTED_AVERAGE; error_mode may be
    Data.INDEPENDENT, Data.DEPENDENT or Data.NO_ERRORS */
public Data unary(int operation, MathType new_type, int sampling_mode,
                  int error_mode)
       throws VisADException, RemoteException;

/** like previous signature of unary, except the result takes
    the MathType of this unless the default Units of that MathType
    conflict with Units of the result, in which case a generic
    MathType with appropriate Units is constructed */
public Data unary(int operation, int sampling_mode, int error_mode)
       throws VisADException, RemoteException;

/** clone this Data object except give it new_type */
public Data changeMathType(MathType new_type)
       throws VisADException, RemoteException;

public Data abs(int sampling_mode, int error_mode)
       throws VisADException, RemoteException;

/** use Data.NEAREST_NEIGHBOR and Data.NO_ERRORS */
public Data abs() throws VisADException, RemoteException;

public Data acos(int sampling_mode, int error_mode)
       throws VisADException, RemoteException;

/** use Data.NEAREST_NEIGHBOR and Data.NO_ERRORS */
public Data acos() throws VisADException, RemoteException;

/** similar methods are defined for the following unary operators:
```

```
acosDegrees, asin, asinDegrees, atan, atanDegrees, ceil, cos,
cosDegrees, exp, floor, log, rint, round, sin, sinDegrees,
sqrt, tan, tanDegrees, negate */
```

3.2.7 Real Methods

A Real object may only be local. Binary operations may be performed between a Real and any other Data object that does not contain Text components; such operations are applied independently with each Real component. Generally useful Real methods include:

```
public final double getValue();

/** get double value converted to unit */
public final double getValue(Unit unit) throws VisADException;

public Unit getUnit();

public ErrorEstimate getError();
```

3.2.8 Text Methods

Text may only be local. The only binary operation that works for Text is Data.ADD, which is interpreted as string concatenation. No unary operations work for Text. Generally useful Text methods include:

```
public String getValue();
```

3.2.9 Tuple Methods

A Tuple object may only be local. Generally useful Tuple methods include:

```
/** return number of components */
public int getDimension();

/** return component for index between 0 and getDimension() - 1 */
public MathType getComponent(int index) throws VisADException;
```

3.2.10 RealTuple Methods

A RealTuple object may only be local. Generally useful RealTuple methods include:

```
/** get Units of Real components */
public Unit[] getTupleUnits();

/** get ErrorEstimates of Real components */
public ErrorEstimate[] getErrors() throws VisADException;

/** get CoordinateSystem */
public CoordinateSystem getCoordinateSystem();
```

3.2.11 Function Methods

A Function object may be either local or remote, a FunctionImpl object may only be local and a RemoteFunction object may only be remote (see Section 6 for more information). Generally useful Function methods are listed below. Note in particular the *resample* method which is invoked implicitly for many visualization and mathematical

operations on Functions and can be invoked by applications for image remapping and a variety of similar Function operations.

```
  /** get dimension of Function domain */
  public int getDomainDimension()
         throws VisADException, RemoteException;

  /** get Units of domain Real components */
  public Unit[] getDomainUnits()
         throws VisADException, RemoteException;

  /** get domain CoordinateSystem */
  public CoordinateSystem getDomainCoordinateSystem()
         throws VisADException, RemoteException;

  /** evaluate Function at domain_value, for 1-D domains */
  public Data evaluate(Real domain_value, int sampling_mode,
                       int error_mode)
         throws VisADException, RemoteException;

  /** evaluate Function at domain_value, for 1-D domains,
      using Data.NEAREST_NEIGHBOR and Data.NO_ERRORS */
  public Data evaluate(Real domain_value)
         throws VisADException, RemoteException;

  /** evaluate Function at domain_value */
  public Data evaluate(RealTuple domain_value, int sampling_mode,
                       int error_mode)
         throws VisADException, RemoteException;

  /** evaluate Function at domain_value using
      Data.NEAREST_NEIGHBOR and Data.NO_ERRORS */
  public Data evaluate(RealTuple domain_value)
         throws VisADException, RemoteException;

  /** return a Field of Function values at samples in set;
      this combines unit conversions, coordinate transforms,
      resampling and interpolation */
  public Field resample(Set set, int sampling_mode, int error_mode)
         throws VisADException, RemoteException;

  /** return the derivative of this Function with respect to d_partial;
      d_partial may occur in this Function's domain RealTupleType, or,
      if the domain has a CoordinateSystem, in its Reference
      RealTupleType; propogate errors according to error_mode */
  public abstract Function derivative(RealType d_partial,
         int error_mode) throws VisADException, RemoteException;

  /** return the derivative of this Function with respect to d_partial;
      set result MathType to derivType; d_partial may occur in this
      Function's domain RealTupleType, or, if the domain has a
      CoordinateSystem, in its Reference RealTupleType;
      propogate errors according to error_mode */
  public abstract Function derivative(RealType d_partial,
         MathType derivType, int error_mode)
         throws VisADException, RemoteException;

  /** return the tuple of derivatives of this Function with respect to
      all RealType components of its domain RealTupleType;
```

```
    propogate errors according to error_mode */
  public abstract Data derivative(int error_mode)
         throws VisADException, RemoteException;

  /** return the tuple of derivatives of this Function with respect
      to all RealType components of its domain RealTupleType;
      set result MathTypes of tuple components to derivType_s;
      propogate errors according to error_mode */
  public abstract Data derivative(MathType[] derivType_s,
         int error_mode) throws VisADException, RemoteException;

  /** return the tuple of derivatives of this Function with respect
      to the RealTypes in d_partial_s; the RealTypes in d_partial_s
      may occur in this Function's domain RealTupleType, or, if the
      domain has a CoordinateSystem, in its Reference RealTupleType;
      set result MathTypes of tuple components to derivType_s;
      propogate errors according to error_mode */
  public abstract Data derivative(RealTuple location,
         RealType[] d_partial_s, MathType[] derivType_s, int error_mode)
         throws VisADException, RemoteException;
```

3.2.12 Field Methods

A Field object may be either local or remote, a FieldImpl object may only be local and a RemoteField object may only be remote (see Section 6 for more information). Generally useful Field methods include:

```
  /** set the values of the Field (at the domain Set samples)
      using the values in range (the length of range must
      equal the length of the domain Set);
      make copies of range values if copy is true */
  public void setSamples(Data[] range, boolean copy)
         throws VisADException, RemoteException;

  /** get the domain Set */
  public Set getDomainSet()
         throws VisADException, RemoteException;

  /** get the Units of the Real components of the domain Set */
  public Unit[] getDomainUnits()
         throws VisADException, RemoteException;

  /** get the CoordinateSystem of the domain Set */
  public CoordinateSystem getDomainCoordinateSystem()
         throws VisADException, RemoteException;

  /** get the Field value at the index-th sample in the
      domain Set */
  public Data getSample(int index)
         throws VisADException, RemoteException;

  /** get the 'Flat' components of this Field's range values
      in their default range Units (as defined by the range of
      the Field's FunctionType); if the range type is a RealType
      it is a 'Flat' component, if the range type is a TupleType
      its RealType components and RealType components of its
      RealTupleType components are all 'Flat' components; the
      return array is dimensioned:
      double[number_of_flat_components][number_of_range_samples] */
```

```
public double[][] getValues()
       throws VisADException, RemoteException;

/** set Field value at the index-th sample in the
    domain Set, to range */
public void setSample(int index, Data range)
       throws VisADException, RemoteException;

/** set Field value at the sample in the domain Set nearest
     domain, to range */
public void setSample(RealTuple domain, Data range)
       throws VisADException, RemoteException;

/** return an Enumeration of RealTuple values in domain Set */
public Enumeration domainEnumeration()
       throws VisADException, RemoteException;

/** return true is this is a FlatField */
public boolean isFlatField();

/** assumes the range type of this is a Tuple and returns
    a Field with the same domain as this, but whose range
    samples consist of the specified Tuple component of the
    range samples of this; in shorthand, this[].component */
public Field extract(int component)
       throws VisADException, RemoteException;

/** combine domains of two outpost nested Fields into a single
    domain and Field; for examples transform the MathType
    (a -> ((b, c) -> d)) into ((a, b, c) -> d) */
public Field domainMultiply()
       throws VisADException, RemoteException;

/** factor Field domain into domains of two nested Fields (with
    factor as outer domain); for examples transform the MathType
    ((a, b, c) -> d) into (a -> ((b, c) -> d)) (where factor = a) */
public Field domainFactor(RealType factor)
       throws VisADException, RemoteException;
```

3.2.13 FieldImpl Method

This describes a single static method of FieldImpl:

```
/** resample all elements of the fields array to the domain
    set of fields[0], then return a Field whose range samples
    are Tuples merging the corresponding range samples from
    each element of fields; if the range of fields[i] is a
    Tuple without a RangeCoordinateSystem, then each Tuple
    component of a range sample of fields[i] becomes a
    Tuple component of a range sample of the result -
    otherwise a range sample of fields[i] becomes a Tuple
    component of a range sample of the result; this assumes
    all elements of the fields array have the same domain
    dimension */
public static Field combine(Field[] fields)
       throws VisADException, RemoteException;
```

3.2.14 Application Example: Synthesizing Fields

In this example we assume that:

```
grid_type =
  ((row, column, level) -> (temperature, pressure, water_vapor))
```

and:

```
vis5d_type = (time -> grid_type)
```

These are the types appropriate for Vis5D data sets synthesized by the example in Section 3.1.14. This example includes constructors for an Integer3DSet and an Integer1DSet, which are described in detail in Section 3.5.3.3, and a constructor for a FlatField, which is an efficient sub-class of FieldImpl described in Section 3.9. The Integer3DSet is an integer lattice of 50 by 50 by 20 points for a Vis5D grid, and the Integer1DSet is a sequence of hour values from 0 to 23. FlatField includes a version of the *setSamples* method that takes an array of floats, in addition to the version of *setSamples* inherited from FieldImpl that takes an array of Data objects. Here's a sample of code for synthesizing a FieldImpl appropriate for a Vis5D data set:

```
// construct an integer 3-D grid
Set grid_set = new Integer3DSet(50, 50, 20);

// construct a sequence of 24 hours
Set time_set = new Integer1DSet(24);

// construct a FieldImpl for a time sequence of grids
FieldImpl vis5d = new FieldImpl(vis5d_type, time_set);

for (int i=0; i<24; i++) {
  // conbstruct a FlatField for the i-th time step
  FlatField grid = new FlatField(grid_type, grid_set);

  // construct an array to hold the gridded field values;
  // data[0] is an array of temperatures, data[1] an array
  // of pressures, and data[2] an array of water_vapors
  float[][] data = new float[3][50 * 50 * 20];

  // ... code to set data values ...

  // set the data values into the grid
  grid.setSamples(data);

  // set grid as the i-th time sample of vis5d
  vis5d.setSample(i, grid);
}
```

3.3 Units

The Unit class defines units for Real values in terms of a user-extensible list of BaseUnits and associated physical quantities. The system-intrinsic list is:

```
ampere    electric current
candela   luminous intensity
kelvin    temperature
kilogram  mass
```

```
meter     length
second    time
mole      amount of substance
radian    angle
```

A Unit is defined by a set of BaseUnits with associated integer exponents, plus a real coefficient and offset. For example, yard = 0.9144 x meter, fahrenheit = (1 / 1.8) x kelvin + 459.67, and joule = kilogram x meter x second^(-2). Two Units are convertible if they have the same set of BaseUnits and integer exponents, or if the exponents of one are negatives of the exponents of the other.

Units with non-zero offsets are dangerous. For example, the conversion of fahrenheit temperature differences to kelvin differences is not correct unless the offset is ignored. In order to avoid this problem, arithmetic operations implicitly convert all inputs to Units with zero offsets.

3.3.1 Unit Methods

Unit is abstract and serializable. A Unit object can only be local (see Section 6 for more information). Its subclasses are all immutable. Applications do not invoke Unit constructors explicitly. Rather they derive new Units be invoking methods of existing Units, or they create new BaseUnits by invoking a static factory method in BaseUnit. Generally useful Unit methods include:

```
/** create a new Unit by raising this (which may not include
    an offset) to power */
public Unit pow(int power) throws UnitException;

/** create a new Unit by multiplication by amount;
    for example, Unit yard = meter.scale(0.9144); */
public Unit scale(double amount) throws UnitException;

/** create a new Unit by adding offset;
    for example, Unit celsius = kelvin.shift(273.15); */
public Unit shift(double offset) throws UnitException;

/** create a new Unit by multiplying this (which may not
    include an offset) by that */
public Unit multiply(Unit that) throws UnitException;

/** create a new Unit by dividing this (which may not
    include an offset) by that */
public Unit divide(Unit that) throws UnitException;
```

3.3.2 SI Variables

The system intrinsic BaseUnits are defined in the SI class as follows:

```
BaseUnit SI.ampere;
BaseUnit SI.candela;
BaseUnit SI.kelvin;
BaseUnit SI.kilogram;
BaseUnit SI.meter;
BaseUnit SI.second;
BaseUnit SI.mole;
BaseUnit SI.radian;
```

3.3.3 BaseUnit Methods

Generally useful BaseUnit methods include:

```
/** create a new BaseUnit with the given quantityName and
    unitName */
public static BaseUnit addBaseUnit(String quantityName,
                       String unitName) throws UnitException;

/** return any baseUnit created in this JVM with the given
    unitName */
public static baseUnit unitNameToUnit(String unitName)

/** return any baseUnit created in this JVM with the given
    quantityName */
public static baseUnit quantityNameToUnit(String quantityName)
```

3.3.4 CommonUnit Variables

The CommonUnit class defines commonly used Units, including:

```
Unit CommonUnit.degree;
Unit CommonUnit.radian;
Unit CommonUnit.second;
/** all BaseUnits have exponent zero in dimensionless */
Unit CommonUnit.dimensionless;
/** promiscuous is compatible with any Unit; useful for constants;
    not the same as null Unit, which is only compatible with
    other null Units */
Unit CommonUnit.promiscuous;
```

3.4 CoordinateSystems

CoordinateSystem is an abstract class whose sub-classes define invertable transformations of the form **R^**n <---> **R^**n between values of various RealTupleTypes. A CoordinateSystem always refers to its reference RealTupleType. On the other hand, a RealTupleType might or might not refer to a default CoordinateSystem. Consequently, a RealTupleType can be one of three kinds with respect to CoordinateSystems:

1. Reference: the RealTupleType doesn't refer to a default CoordinateSystem but a CoordinateSystem refers to the RealTupleType.
2. Equivalent: the RealTupleType refers to a default CoordinateSystem and, thus, refers indirectly to a reference RealTupleType.
3. Uninvolved: the RealTupleType neither refers to a default CoordinateSystem nor is referred to by a CoordinateSystem.

Thus CoordinateSystems define equivalence classes of those RealTupleTypes with the same reference. For example, (polar_sterographic_row, polar_sterographic_column), (lambert_conformal_row, lambert_conformal_column) and other map projections could form an equivalence class relative to, and including, the Reference (latitude, longitude). Each of the map projections would include a default CoordinateSystem that defined its mathematical transformation between (row, column) and (latitude, longitude).

The default CoordinateSystem defined by a RealTupleType can be over-ridden for RealTuple values of that type, in order to support data-dependent CoordinateSystems. For example, meteorologists use (latitude, longitude, pressure) as a CoordinateSystem with Reference (latitude, longitude, altitude), where the mathematical transformation can vary depending on the vertical distribution of pressures. A default CoordinateSystem can only be over-ridden by a CoordinateSystem with the same Reference.

3.4.1 CoordinateSystem Constructors

CoordinateSystem is abstract and serializable. A CoordinateSystem object can only be local (see Section 6 for more information). Applications generally do not invoke CoordinateSystem methods, but they construct new CoordinateSystem objects and define new CoordinateSystem subclasses.

Note that care should be taken to make sure that:

1. The order of RealType components in a reference RealTupleType is consistent with the computations of the *toReference* and *fromReference* methods.
2. The Units of the RealType components in a reference RealTupleType are consistent with the values assumed by the *toReference* and *fromReference* methods.
3. The order of RealType components of a RealTupleType with a CoordinateSystem is consistent with the computations of the *toReference* and *fromReference* methods.

The constructor for the abstract CoordinateSystem class is:

```
/** user-defined subclasses must supply reference and units */
public CoordinateSystem(RealTupleType reference, Unit[] units)
       throws VisADException;
```

Constructors for specific CoordinateSystems included with VisAD include:

```
/** construct a CoordinateSystem for (latitude, longitude,
    radius) relative to a 3-D Cartesian reference;
    this constructor supplies units =
    {CommonUnit.Degree, CommonUnit.Degree, null} to the super
    constructor, in order to ensure Unit compatibility with its
    use of trigonometric functions */
public SphericalCoordinateSystem(RealTupleType reference)
       throws VisADException;

/** construct a CoordinateSystem for (longitude, radius)
    relative to a 2-D Cartesian reference;
    this constructor supplies units = {CommonUnit.Degree, null}
    to the super constructor, in order to ensure Unit
    compatibility with its use of trigonometric functions */
public PolarCoordinateSystem(RealTupleType reference)
       throws VisADException;

/** construct a CoordinateSystem that whose transforms invert
    the transforms of inverse (i.e., toReference and
    fromReference are switched); for example, this could be
    used to define Cartesian coordinates releative to a
    refernce in spherical coordinates */
```

```
public InverseCoordinateSystem(RealTupleType reference,
                               CoordinateSystem inverse)
       throws VisADException;

/** construct a CoordinateSystem for grid coordinates (e.g.,
    (row, column, level) in 3-D) relative to the value space
    of set; for example, if satellite pixel locations are
    defined by explicit latitudes and longitude, these could
    be used to construct a Gridded2DSet which could then be
    used to construct a GridCoordinateSystem for (ImageLine,
    ImageElement) coordinates relative to reference coordinates
    (Latitude, Longitude) */
public GridCoordinateSystem(GriddedSet set)
       throws VisADException;
```

3.4.2 CoordinateSystem Methods

Extensions of CoordinateSystem must implement the following methods:

```
/** convert RealTuple values to Reference coordinates;
    for efficiency, input and output values are passed as
    double[][] arrays rather than RealTuple[] arrays; the
    array indexes are:
    double[tuple_dimension][number_of_tuples] */
public double[][] toReference(double[][] tuples)
       throws VisADException;

/** convert RealTuple values from Reference coordinates */
public double[][] fromReference(double[][] tuples)
       throws VisADException;
```

The following methods are implemented in CoordinateSystem in terms of the above methods, but for efficiency's sake extensions of CoordinateSystem may override those with direct implementations:

```
public float[][] toReference(float[][] tuples)
       throws VisADException;

public float[][] fromReference(float[][] tuples)
       throws VisADException;
```

3.5 Sets

A Field object approximates a function by interpolating its values at a finite subset of its domain [3]. A Field object includes a Set object that defines the finite sampling of the function's domain. This Set object also defines the CoordinateSystem of the Field's domain and the Units of the domain's RealType components. The Set class has many sub-classes for different ways of defining finite subsets of the Set's domain **R^**n (*n* is called the domain dimension of the Set). A partial Set class hierarchy is:

```
Set
  SimpleSet
     DoubleSet
     FloatSet
     SampledSet
       ProductSet
```

```
UnionSet
GriddedSet
  LinearNDSet
    IntegerNDSet
  Gridded1DSet
    Linear1DSet
      Integer1DSet
  Gridded2DSet
    Linear2DSet
      LinearLatLonSet
      Integer2DSet
  Gridded3DSet
    Linear3DSet
      Integer3DSet
  Gridded1DDoubleSet
IrregularSet
  Irregular1DSet
  Irregular2DSet
  Irregular3DSet
```

A SimpleSet is embedded on a sub-manifold of dimension *m* in **R^**n (*m* is called the manifold dimension of the Set). A DoubleSet with domain dimension is just the large but finite set of values in **R^**n representable by *n* IEEE double precision floating point values. Similarly for FloatSet and single precision. The SampledSet class implements some common methods for its subclasses. The samples of a GriddedSet are organized in an *m*-dimensional grid. For a LinearSet this grid is aligned to the axes of the domain **R^**n and for an IntegerSet the grid points form an integer lattice based at the origin. The samples of an IrregularSet are not organized. ProductSets and UnionSets allow Sets to be defined as products and unions of other Sets.

Note that Set is a sub-class of Data, so Sets are full-fledged Data objects in addition to being a form of metadata for Fields. For example, a set of map boundaries would be a Set with domain dimension $n = 2$ and manifold dimension $m = 1$.

Note also that there is a Set class in the java.util package as of JDK 1.2. Thus applications should avoid combining 'import java.*;' with 'import java.util.*;'.

3.5.1 Defining Interpolation Algorithms by Extending the Set Class

The resample method of the Field class is the workhorse of the system. It takes a Set as an argument and returns a new Field containing values of the original Field sampled at the Set locations. It also does any necessary Unit conversions and CoordinateSystem transformations. The resample method is invoked implicitly whenever needed for mathematical and visualization operations involving Fields. The resample method includes options to interpolate Field values by either nearest neighbor or weighted average. Any degree polynomial interpolation, single stage Barnes and Cressman analyses, and a wide variety of other interpolation schemes can be expressed as weighted averages. Fields get weights from the *valueToInterp* method of SimpleSet. Thus developers may implement new interpolation algorithms by extending the Set class.

Implementation of interpolation methods not consistent with weighted average would require extensions of Field and FlatField. Nearest neighbor resampling uses the *valueToIndex* method of Set.

The *getWedge* method of SimpleSet is important for the efficiency of Field resampling and interpolation. The samples of one Set are passed to the *valueToInterp* and *valueToIndex* of another set in an order defined the first Set's *getWedge* method. Sets use *getWedge* to define a spatially coherent order of their samples. It is important that developers who extend SimpleSet try to define spatially coherent orders in their implementations of *getWedge*.

Note that *valueToInterp* and *valueToIndex* generally throw an Exception for any Set whose manifold dimension is less than its domain dimension. Thus the *resample* method does not work for Fields whose domain Sets have manifold dimension less than their domain dimension. In order to resample a Field X over a domain of dimension N with manifold dimension M < N, applications must explicitly copy values of X to another Field Y whose domain has dimension M and is a parameterization of the sub-manifold containing the samples of X. For example, if N = 3 and M = 2, then the samples of X lie on a 2-D surface embedded in a 3-D space, and the domain of Y should be a parameterization of this surface, with samples locations corresponding to X's sample locations on the surface.

3.5.2 The Delaunay Class for Irregular Sets

The topology of IrregularSets is recorded, and in some cases computed, in the Delaunay classes, which form the following hierarchy:

```
Delaunay
  DelaunayClarkson
  DelaunayWatson
  DelaunayFast
  DelaunayCustom
```

The DelaunayClarkson class computes Delaunay triangulations in any dimension between 2 and 8 using Ken Clarkson's algorithm. DelaunayCustom constructors accept sampling topologies from applications. The DelaunayWatson class computes Delaunay triangulations in 2 or 3 dimensions using David Watson's algorithm. The DelaunayFast class computes non-Delaunay triangulations quickly.

Note that any computation of Delaunay or approximate Delaunay topology is extremely slow and apt to exceed available memory for large Sets. Hence, where an irregular topology is known to the application, we strongly recommend that the topology be supplied by the application through the DelaunayCustom constructor.

3.5.3 Set Constructors

Set is a subclass of DataImpl. A Set object may only be local. The Set classes include the following constructors:

3.5.3.1 DoubleSet and FloatSet Constructors

These are the finite but very large sets of values representable with N IEEE floats or doubles. Because of their size, they may not be used as Field domains. They are primarily used (with N = 1) for FlatField range values, where they cause range values to be stored in IEEE floats or doubles.

```
/** the set of values representable by N doubles;
```

```
    type must be a RealType, a RealTupleType or a SetType;
    coordinate_system and units must be compatible with defaults
    for type, or may be null;
    a DoubleSet may not be used as a Field domain */
public DoubleSet(MathType type, CoordinateSystem coordinate_system,
                 Unit[] units) throws VisADException;

/** the set of values representable by N floats;
    type must be a RealType, a RealTupleType or a SetType;
    coordinate_system and units must be compatible with defaults
    for type, or may be null;
    a FloatSet may not be used as a Field domain */
public FloatSet(MathType type, CoordinateSystem coordinate_system,
                Unit[] units) throws VisADException;
```

3.5.3.2 LinearSet Constructors

LinearSet is an interface implemented by Linear1DSet, Linear2DSet, Linear3DSet and LinearNDSet. Linear1DSets are finite arithmetic progressions of values. Higher dimensional LinearSets are product sets of Linear1DSets. All LinearSets have manifold dimension equal to their domain dimension, although any of the component Linear1DSets may consist of a single sample (in this case, the *valueToIndex* and *valueToInterp* methods will throw an Exception).

Linear1DSet, Linear2DSet, Linear3DSet are redundant with LinearNDSet but have more efficient implementations.

The samples of a LinearSet are in raster order, with component values for the first dimension changing fastest and component values for the last dimension changing slowest (this is the same as the ordering of elements in a multi-dimensional Fortran array). For example, given a Linear2DSet with domain type (X, Y) that is a product of six X samples and five Y samples, the 2-D samples are ordered as:

```
           Y (second) component

     X      0   6  12  18  24
            1   7  13  19  25
  (first)   2   8  14  20  26
            3   9  15  21  27
component   4  10  16  22  28
            5  11  17  23  29
```

LinearSets extend GriddedSets, described in Section 3.5.3.3. GriddedSets have rectangular topology while LinearSets have rectangular topology and geometry.

```
/** an arithmetic progression of length values between first and last;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Linear1DSet(MathType type,
                   double first, double last, int length,
                   CoordinateSystem coordinate_system, Unit[] units,
                   ErrorEstimate[] errors) throws VisADException;

/** a 1-D arithmetic progression with null errors and generic type */
public Linear1DSet(double first, double last, int length)
       throws VisADException;
```

```
/** a 2-D cross product of arithmetic progressions;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Linear2DSet(MathType type,
                   double first1, double last1, int length1,
                   double first2, double last2, int length2,
                   CoordinateSystem coordinate_system, Unit[] units,
                   ErrorEstimate[] errors) throws VisADException;

/** a 2-D cross product of arithmetic progressions with
    null errors and generic type */
public Linear2DSet(double first1, double last1, int length1,
                   double first2, double last2, int length2)
       throws VisADException;

/** a 3-D cross product of arithmetic progressions;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Linear3DSet(MathType type,
                   double first1, double last1, int length1,
                   double first2, double last2, int length2,
                   double first3, double last3, int length3,
                   CoordinateSystem coordinate_system, Unit[] units,
                   ErrorEstimate[] errors) throws VisADException;

/** a 3-D cross product of arithmetic progressions with
    null errors and generic type */
public Linear3DSet(double first1, double last1, int length1,
                   double first2, double last2, int length2,
                   double first3, double last3, int length3)
       throws VisADException;

/** a 2-D cross product of arithmetic progressions that whose east
    and west edges may be joined (for interpolation purposes);
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public LinearLatLonSet(MathType type,
                       double first1, double last1, int length1,
                       double first2, double last2, int length2,
                       CoordinateSystem coordinate_system,
                       Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 2-D cross product of arithmetic progressions that whose east
    and west edges may be joined (for interpolation purposes), with
    null errors, CoordinateSystem and Units are defaults from type */
public LinearLatLonSet(MathType type,
                       double first1, double last1, int length1,
                       double first2, double last2, int length2)
       throws VisADException;

/** construct an N-dimensional set as the product of N Linear1DSets;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public LinearNDSet(MathType type, Linear1DSet[] sets,
                   CoordinateSystem coordinate_system,
                   Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** construct an N-dimensional set as the product of N Linear1DSets,
```

```
    with null errors, CoordinateSystem and Units are defaults from
    type */
  public LinearNDSet(MathType type, Linear1DSet[] sets)
         throws VisADException;

  /** construct an N-dimensional set as the product of N arithmetic
    progressions; coordinate_system and units must be compatible
    with defaults for type, or may be null; errors may be null */
  public LinearNDSet(MathType type, double[] firsts, double[] lasts,
                     int[] lengths, CoordinateSystem coordinate_system,
                     Unit[] units, ErrorEstimate[] errors)
         throws VisADException;

  /** construct an N-dimensional set as the product of N arithmetic
    progressions, with null errors, CoordinateSystem and Units are
    defaults from type */
  public LinearNDSet(MathType type, double[] firsts, double[] lasts,
                     int[] lengths) throws VisADException;
```

3.5.3.3 IntegerSet Constructors

IntegerSet is an interface implemented by Integer1DSet, Integer2DSet, Integer3DSet and IntegerNDSet. These classes are simple extensions of the corresponding LinearSet classes that constrain arithmetic progressions to sequences of consecutive integers based at zero. Integer1DSet, Integer2DSet, Integer3DSet are redundant with IntegerNDSet but have more efficient implementations.

IntegerSets are useful as the domains of Fields that are really just simple 1-D, 2-D, 3-D or N-D arrays of values.

```
  /** construct a 1-dimensional set with values {0, 1, ..., lengthX-1};
    coordinate_system and units must be compatible with defaults for
    type, or may be null; errors may be null */
  public Integer1DSet(MathType type, int lengthX,
                      CoordinateSystem coordinate_system,
                      Unit[] units, ErrorEstimate[] errors)
         throws VisADException;

  /** a 1-D set with null errors and generic type */
  public Integer1DSet(int lengthX)
         throws VisADException;

  /** construct a 2-dimensional set with values
    {0, 1, ..., lengthX-1} x {0, 1, ..., lengthY-1};
    coordinate_system and units must be compatible with defaults for
    type, or may be null; errors may be null */
  public Integer2DSet(MathType type, int lengthX, lengthY,
                      CoordinateSystem coordinate_system,
                      Unit[] units, ErrorEstimate[] errors)
         throws VisADException;

  /** a 2-D set with null errors and generic type */
  public Integer2DSet(int lengthX, lengthY)
         throws VisADException;

  /** construct a 3-dimensional set with values {0, 1, ..., lengthX-1}
    x {0, 1, ..., lengthY-1} x {0, 1, ..., lengthZ-1};
    coordinate_system and units must be compatible with defaults for
    type, or may be null; errors may be null */
```

```
public Integer3DSet(MathType type, int lengthX, lengthY, lengthZ,
                    CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 3-D set with null errors and generic type */
public Integer3DSet(int lengthX, lengthY, lengthZ)
       throws VisADException;

/** construct an N-dimensional set with values in the cross product
    of {0, 1, ..., lengths[i]-1}
    for i=0, ..., lengths[lengths.length-1];
    coordinate_system and units must be compatible with defaults for
    type, or may be null; errors may be null */
public IntegerNDSet(MathType type, int[] lengths,
                    CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** an N-D set with null errors and generic type */
public IntegerNDSet(int[] lengths)
       throws VisADException;
```

3.5.3.4 GriddedSet Constructors

GriddedSets are N-dimensional sets with rectangular topologies but not necessarily rectangular geometries. GriddedSet implements the general N-dimensional case (although that implementation is not complete in the initial release) and is extended by Gridded1DSet, Gridded2DSet and Gridded3DSet, which are complete.

GriddedSets may have manifold dimension less than (or equal to) their domain dimension. A GriddedSet with domain dimension N and manifold dimension M defines an M-dimensional grid of samples embedded in an N-dimensional space. In the GriddedSet constructors, the arguments lengthX, lengthY and lengthZ define the numbers of samples along each dimension of the grid (so the number of length arguments defines the manifold dimension), and the samples array argument defines the locations of grid points in N-dimensional domain space. The samples array has type float[][] with dimensions float[N][number_of_samples]. Thus the i-th point in the grid is located at:

```
(samples[0][i], samples[1][i], ..., samples[N-1][i]).
```

The samples are in raster order, with the first grid dimension changing fastest and the last grid dimension changing slowest. That is, the first lengthX samples form the first 'column' of the grid, the first (lengthX * lengthY) samples for the first sub-plane of the grid, and so on.

If the manifold dimension is less than the domain dimension or any of the grid sizes (i.e., lengthX, lengthY or lengthZ) is 1, then the *valueToIndex* and *valueToInterp* methods will throw an Exception. If the manifold dimension equals the domain dimension and all of the grid sizes is greater than 1, then the GriddedSet constructor will perform numerical checks on the samples array to ensure that form a valid grid (e.g., to ensure that they are sorted in the 1-D case).

```
/** a 1-D sorted sequence with no regular interval; samples array
```

```
    is organized float[1][number_of_samples] where lengthX =
    number_of_samples; samples must be sorted (either increasing
    or decreasing); coordinate_system and units must be compatible
    with defaults for type, or may be null; errors may be null */
public Gridded1DSet(MathType type, float[][] samples, int lengthX,
                    CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 1-D sequence with no regular interval with null errors,
    CoordinateSystem and Units are defaults from type */
public Gridded1DSet(MathType type, float[][] samples, int lengthX)
       throws VisADException;

/** a 1-D sorted sequence with no regular interval; samples array
    is organized double[1][number_of_samples] where lengthX =
    number_of_samples; samples must be sorted (either increasing
    or decreasing); coordinate_system and units must be compatible
    with defaults for type, or may be null; errors may be null */
    Gridded1DDoubleSet is useful for sequences of DataTime values
    represented as double precision seconds */
public Gridded1DDoubleSet(MathType type, double[][] samples,
                    int lengthX, CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 1-D sequence with no regular interval with null errors,
    CoordinateSystem and Units are defaults from type;
    Gridded1DDoubleSet is useful for sequences of DataTime values
    represented as double precision seconds */
public Gridded1DDoubleSet(MathType type, double[][] samples,
                          int lengthX)
       throws VisADException;

/** a 2-D set whose topology is a lengthX x lengthY grid;
    samples array is organized float[2][number_of_samples] where
    lengthX * lengthY = number_of_samples; samples must form a
    non-degenerate 2-D grid (no bow-tie-shaped grid boxes); the
    X component increases fastest in the second index of samples;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Gridded2DSet(MathType type, float[][] samples, int lengthX,
                    int lengthY, CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 2-D set whose topology is a lengthX x lengthY grid, with
    null errors, CoordinateSystem and Units are defaults from type */
public Gridded2DSet(MathType type, float[][] samples, int lengthX,
                    int lengthY) throws VisADException;

/** a 2-D set with manifold dimension = 1; samples array is
    organized float[2][number_of_samples] where lengthX =
    number_of_samples; no geometric constraint on samples;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Gridded2DSet(MathType type, float[][] samples, int lengthX,
                    CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;
```

```
/** a 2-D set with manifold dimension = 1, with null errors,
    CoordinateSystem and Units are defaults from type */
public Gridded2DSet(MathType type, float[][] samples, int lengthX)
       throws VisADException;

/** a 3-D set whose topology is a lengthX x lengthY x lengthZ
    grid; samples array is organized float[3][number_of_samples]
    where lengthX * lengthY * lengthZ = number_of_samples;
    samples must form a non-degenerate 3-D grid (no bow-tie-shaped
    grid cubes);  the X component increases fastest and the Z
    component slowest in the second index of samples;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Gridded3DSet(MathType type, float[][] samples, int lengthX,
                    int lengthY, int lengthZ,
                    CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 3-D set whose topology is a lengthX x lengthY x lengthZ
    grid, with null errors, CoordinateSystem and Units are
    defaults from type */
public Gridded3DSet(MathType type, float[][] samples, int lengthX,
                    int lengthY, int lengthZ) throws VisADException;

/** a 3-D set with manifold dimension = 2; samples array is
    organized float[3][number_of_samples] where lengthX * lengthY
    = number_of_samples; no geometric constraint on samples; the
    X component increases fastest in the second index of samples;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Gridded3DSet(MathType type, float[][] samples, int lengthX,
                    int lengthY, CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 3-D set with manifold dimension = 2, with null errors,
    CoordinateSystem and Units are defaults from type */
public Gridded3DSet(MathType type, float[][] samples, int lengthX,
                    int lengthY) throws VisADException;

/** a 3-D set with manifold dimension = 1; samples array is
    organized float[3][number_of_samples] where lengthX =
    number_of_samples; no geometric constraint on samples;
    coordinate_system and units must be compatible with defaults
    for type, or may be null; errors may be null */
public Gridded3DSet(MathType type, float[][] samples, int lengthX,
                    CoordinateSystem coordinate_system, Unit[] units,
                    ErrorEstimate[] errors)
       throws VisADException;

/** a 3-D set with manifold dimension = 1, with null errors,
    CoordinateSystem and Units are defaults from type */
public Gridded3DSet(MathType type, float[][] samples, int lengthX)
       throws VisADException;
```

3.5.3.5 IrregularSet Constructors

IrregularSets are N-dimensional sets with irregular topologies consisting of lists of (N+1)-gons (i.e., line segments in 1 dimension, triangles in 2 dimensions, tetrahedra in 3 dimensions, etc). IrregularSet implements the general N-dimensional case (although that implementation is not complete in the initial release) and is extended by Irregular1DSet, Irregular2DSet and Irregular3DSet, which are complete.

The samples array argument to the IrregularSet constructors defines the locations of sample points in N-dimensional domain space. The samples array has type float[][] with dimensions float[N][number_of_samples]. Thus the i-th sample point is located at:

```
(samples[0][i], samples[1][i], ..., samples[N-1][i]).
```

IrregularSets may have manifold dimension less than or equal to their domain dimension. If the manifold dimension is less than the domain dimension, then the *valueToIndex* and *valueToInterp* methods throw Exceptions.

In 1 dimension the topology is constructed merely by sorting the samples. In higher dimensions the topology may be constructed by a Delaunay triangulation or may be specified in the constructor (using the DelaunayCustom class). See Section 3.5.5 for more information about Delaunay classes.

```
/** a 1-D irregular set; samples array is organized
    float[1][number_of_samples]; samples need not be
    sorted - the constructor sorts samples to define
    a 1-D "triangulation";
    coordinate_system and units must be compatible with
    defaults for type, or may be null; errors may be null */
public Irregular1DSet(MathType type, float[][] samples,
                      CoordinateSystem coordinate_system,
                      Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

/** a 1-D irregular set with null errors, CoordinateSystem
    and Units are defaults from type */
public Irregular1DSet(MathType type, float[][] samples)
       throws VisADException;

/** a 2-D irregular set; samples array is organized
    float[2][number_of_samples];  no geometric constraint on
    samples; if delan is non-null it defines the topology of
    samples (which must have manifold dimension 2), else the
    constructor computes a topology with manifold dimension 2;
    note that Gridded2DSet can be used for an irregular set
    with domain dimension 2 and manifold dimension 1;
    coordinate_system and units must be compatible with
    defaults for type, or may be null; errors may be null */
public Irregular2DSet(MathType type, float[][] samples,
                      CoordinateSystem coordinate_system,
                      Unit[] units, ErrorEstimate[] errors,
                      Delaunay delan)
       throws VisADException;

/** a 2-D irregular set with null errors, CoordinateSystem
    and Units are defaults from type; topology is computed
    by the constructor */
public Irregular2DSet(MathType type, float[][] samples)
```

```
        throws VisADException;

  /** a 3-D irregular set; samples array is organized
      float[3][number_of_samples];  no geometric constraint on
      samples; if delan is non-null it defines the topology of
      samples (which may have manifold dimension 2 or 3), else
      the constructor computes a topology with manifold dimension
      3; note that Gridded3DSet can be used for an irregular set
      with domain dimension 3 and manifold dimension 1;
      coordinate_system and units must be compatible with
      defaults for type, or may be null; errors may be null */
  public Irregular3DSet(MathType type, float[][] samples,
                        CoordinateSystem coordinate_system,
                        Unit[] units, ErrorEstimate[] errors,
                        Delaunay delan)
         throws VisADException;

  /** a 3-D irregular set with null errors, CoordinateSystem
      and Units are defaults from type; topology is computed
      by the constructor */
  public Irregular3DSet(MathType type, float[][] samples)
         throws VisADException;
```

3.5.3.6 ProductSet and UnionSet Constructors

ProductSets are SampledSets that are defined as products of other SampledSets (called the ProductSet's factor sets). The domain dimension of a ProductSet is the sum of the domain dimensions of its factors and similarly its manifold dimension is the sum of the manifold dimensions of its factors. The order of samples in a ProductSet is the rasterization of the orders of samples of its factors. As the index of the ProductSet increases, the index of the first factor varies fastest and the index of the last factor varies slowest.

UnionSets are SampledSets that are defined as unions of other SampledSets. All the sets in the union must have the same domain dimension and they must all have the same manifold dimension. Note that the *valueToInterp* method is not implemented for UnionSets but the *valueToIndex* method is. Thus if a UnionSet is the domain set of a Field, arithmetic operations involving the Field must specify the Data.NEAREST_NEIGHBOR resampling mode rather than Data.WEIGHTED_AVERAGE. The order of samples in a UnionSet is the serialization of the orders of samples of its components. As the index of the UnionSet increases, the samples of the first component are enumerated first and the samples of the last component are enumerated last.

```
  /** create the product of the sets array; coordinate_system
      and units must be compatible with defaults for type,
      or may be null; errors may be null */
  public ProductSet(MathType type, SampledSet[] sets,
                    CoordinateSystem coordinate_system,
                    Unit[] units, ErrorEstimate[] errors)
         throws VisADException;

  /** create the product of the sets array, with null errors,
      CoordinateSystem and Units are defaults from type */
  public ProductSet(MathType type, SampledSet[] sets)
         throws VisADException;

  /** create the union of the sets array; coordinate_system
```

```
    and units must be compatible with defaults for type,
    or may be null; errors may be null */
  public UnionSet(MathType type, SampledSet[] sets,
                  CoordinateSystem coordinate_system,
                  Unit[] units, ErrorEstimate[] errors)
       throws VisADException;

  /** create the union of the sets array, with null errors,
    CoordinateSystem and Units are defaults from type */
  public UnionSet(MathType type, SampledSet[] sets)
       throws VisADException;
```

3.5.4 Set Methods

Applications generally do not invoke Set methods, but they construct new Set objects and may define new Set subclasses. New Set subclasses must either implement or inherit these methods:

```
  /** return an enumeration of sample indices in a spatially
    coherent order; this is useful for efficiency */
  public int[] getWedge();

  /** return an enumeration of sample values in index order
     (i.e., not in getWedge order); the return array is
     organized as float[domain_dimension][number_of_samples] */
  public float[][] getSamples() throws VisADException;

  /** convert an array of indices to an array of sample values;
    the return array is organized as
    float[domain_dimension][indices.length] */
  public float[][] indexToValue(int[] indices) throws VisADException;

  /** convert an array of values to an array of indices of the nearest
    samples; the values array is organized as
    float[domain_dimension][number_of_values] */
  public int[] valueToIndex(float[][] values) throws VisADException;

  /** convert an array of indices to an array of double precision
    sample values; this precision is currently only meaningfulful
    for Linear1DSet and Gridded1DDoubleSet where it is intended
    to represent date/time values as double precision seconds;
    the return array is organized as
    double[domain_dimension][indices.length] */
  public double[][] indexToDouble(int[] indices) throws VisADException;

  /** convert an array of double precision values to an array of
    indices of the nearest samples; this precision is currently
    only meaningfulful for Linear1DSet and Gridded1DDoubleSet
    where it is intended to represent date/time values as double
    precision seconds; the values array is organized as
    double[domain_dimension][number_of_values] */
  public int[] doubleToIndex(double[][] values) throws VisADException;
```

3.5.5 SimpleSet Method

```
  /** convert an array of values to arrays of indices and weights for
    those indices, appropriate for interpolation; the values array is
    organized as float[domain_dimension][number_of_values]; indices
```

```
    and weights must be passed in as int[number_of_values][] and
    float[number_of_values][]; on return, quantity( values[.][i] )
    can be estimated as the sum over j of
    weights[i][j] * quantity (sample at indices[i][j]);
    no estimate possible if indices[i] and weights[i] are null */
public void valueToInterp(float[][] values, int[][] indices,
                          float[][] weights) throws VisADException;
```

3.5.6 Delaunay Constructors

The Delaunay class is serializable. A Delaunay object may only be local. The Delaunay classes include the following useful constructor:

```
/** the DelaunayCustom constructor allows applications to define
    sampling topologies; the samples array is organized as
    float[domain_dimension][number_of_samples] and the tris arrays
    is organized as int[number_of_tris][manifold_dimension + 1];
    each "tri" is a list of sample indices, and is a triangle,
    tetrahedron, etc depending on manifold dimension */
public DelaunayCustom(float[][] samples, int[][] tris)
       throws VisADException;
```

3.6 ErrorEstimates

The ErrorEstimate class contains an estimate of the variance of error associated with a value or a set of values. ErrorEstimates are included with individual Real values, and with each RealType component in the range of FlatFields. For example, one range component of a FlatField may consist of all temperature values in a model output grid, and these would be associated with a single average ErrorEstimate (see Section 3.9).

Data operations include options to propagate ErrorEstimates assuming that errors are distributed either independently or dependently, as well as an option to not propagate ErrorEstimates.

The VisAD ErrorEstimates are not a substitute for a detailed error analysis, but can provide a quick estimate of error magnitude and the possible need for detailed analysis.

3.6.1 ErrorEstimate Constructors

The ErrorEstimate class is serializable. An ErrorEstimate object may only be local. The ErrorEstimate class include the following constructors:

```
/** construct an error distribution of number values with
    given mean and variance, in Unit unit */
public ErrorEstimate(double variance, double mean,
                     long number, Unit unit);

/** construct an error distribution of 1 value with
    given mean and variance, in Unit unit */
public ErrorEstimate(double mean, double variance, Unit unit);
```

3.7 AuditTrails

The AuditTrail class contains an ordered sequence of text strings documenting the history of a Data object, starting with external data sources (e.g., data files and URLs) and including Data operations. In order to conserve memory, AuditTrail objects are only

associated with top-level Data objects (i.e., Data objects that are not components of Fields or Tuples).

The AuditTrail class is not yet implemented, so there is no constructor and method documentation.

3.8 Missing Data

Any Data object or primitive value may be marked as *missing*, meaning that its value is unknown or undefined. Missing values may be generated as the result of sensor failures, arithmetic failures (e.g., division by zero), or to mark incomplete data coverage (e.g., temperatures are not available for one time step of a model output). The NaN (Not a Number) value of the IEEE floating point standard is used to represent missing *floats* and *doubles* in VisAD, since it has the correct arithmetic semantics (e.g., X .OP. NaN = NaN for any value X and any operation .OP.).

3.9 FlatFields - Data Operations and Efficiency

There is a natural trade-off between generality and efficiency, so the generality of the VisAD data model poses a challenge for efficiency. Efficiency is achieved by incorporating the following rule at all levels of the system:

* Apply all data operations to arrays of values rather than individual values, and avoid methods that are invoked once per data value.

The effectiveness of this rule was demonstrated in the C implementation of VisAD [8, 9], which had a general data model like the Java implementation.

The large Data objects in any application are Fields. Most array data in numerical programs are finite samplings of functions (for example, images are finite samplings of continuous radiance functions with a pixel for each sample) and these correspond to Fields. Even arrays that do not correspond to any obvious continuous function can be represented by Fields whose domains are sets of integers from 1 to N. The obvious way to implement the Field class is with an array of range sample objects, which would violate our rule because Field operations invoke methods on each range object. Thus the Field class is extended by FlatField, which simulates an array of range objects with arrays of Java primitive values. A FlatField can be used for a Field under the following two conditions:

1. The MathType of the Field range is a RealType, a RealTupleType, or a TupleType whose components are all RealTypes or RealTupleTypes (this allows subsets of a FlatField's range components to be grouped into RealTupleTypes to document CoordinateSystems).
2. All range samples have identical metadata, including Units, CoordinateSystems, shared ErrorEstimates, etc.

FlatFields are appropriate for images, multi-channel images, multi-variate grids, time series and many other types of numerical data arrays. Complex data may be implemented by Fields whose range samples are FlatFields. For example, a time sequence of images may be

implemented by a Field whose domain is a set of time steps, and whose range samples are each images stored in FlatField.

In addition to computational efficiency, FlatFields also have better storage efficiency than Fields. Java primitive data require less storage than Java objects, shared metadata objects require less total space, and when possible function range values are stored in *bytes*, *shorts* or *ints* rather than *floats*. The FlatField constructor accepts range sampling Sets for each RealType component of its range. If the size of the sampling Set for a range component is ≤ 255, then values for that component are encoded as indices into that Set and stored in an array of *bytes* (the 256th code is used to represent *missing* values). Arrays of *shorts* or *ints* are used for larger set sizes, as appropriate. The default range sampling Sets are 1-D FloatSets, which cause range value to b stored as floats.

Numerical precision problems occur and can be very difficult to diagnose when they do. Thus developers may want to pass DoubleSets to the range sampling Sets argument of the FlatField constructor, in order to avoid precision problems.

Float.NaN and Double.NaN are used to represent missing float and double values. This avoids time-consuming explicit tests for missing values, since these IEEE NaNs have the right arithmetic semantics for missing values.

3.9.1 FlatField Constructors

FlatField is a subclass of FieldImpl. A FlatField object may only be local. The FlatField class include the following constructors:

```
/** FlatField is a sampled function whose range is a Real,
    a RealTuple, or a Tuple of Reals and RealTuples; if range
    is a RealTuple, range_coordinate_system may be non-null
    but must have the same Reference as RangeType default
    CoordinateSystem; domain_set defines the domain sampling;
    range_sets define samplings for range values - if range_set[i]
    is null, the i-th range component values are stored as doubles;
    if range_set[i] is non-null, the i-th range component values are
    stored in bytes if range_sets[i].getLength() < 256, stored in
    shorts if range_sets[i].getLength() < 65536, etc;
    any argument but type may be null */
public FlatField(FunctionType type, Set domain_set,
                 CoordinateSystem range_coordinate_system,
                 Set[] range_sets, Unit[] units)
       throws VisADException;

/** similar to the previous constructor, except that if
    range_coordinate_systems[i] is non-null, then the i-th
    component of the range type must be a RealTupleType whose
    default CoordinateSystem has the same Reference */
public FlatField(FunctionType type, Set domain_set,
                 CoordinateSystem[] range_coordinate_systems,
                 Set[] range_sets, Unit[] units)
       throws VisADException;
```

3.9.2 FlatField Methods

FlatField overrides many of the FieldImpl methods, plus it defines a number of methods for accessing range values as arrays of doubles and floats, and accessing range metadata (which are shared by all range samples).

```
  /** convert FlatField to FieldImpl */
  public Field convertToField()
         throws VisADException, RemoteException;

  /** return array of Units associated with each RealType
      component of range; these may differ from default
      Units of range RealTypes, but must be convertable */
  public Unit[][] getRangeUnits();

  /** return range CoordinateSystem assuming range type is
      a RealTupleType (throws a TypeException if its not);
      this may differ from default CoordinateSystem of
      range RealTupleType, but must be convertable */
  public CoordinateSystem[] getRangeCoordinateSystem();

  /** return range CoordinateSystem associated with
      RealTupleType that is index-th component of range
      TupleType; this may differ from default
      CoordinateSystem of RealTupleType component of
      range TupleType, but must be convertable */
  public CoordinateSystem[] getRangeCoordinateSystem(int index);

  /** return array of ErrorEstimates associated with each
      RealType component of range; each ErrorEstimate is a
      mean error for all samples of a range RealType
      component */
  public ErrorEstimates[] getRangeErrors();

  /** set ErrorEstimates associated with each RealType
      component of range */
  public void setRangeErrors(ErrorEstimates[] errors)
         throws VisADException;

  /** set range array as range values of this FlatField;
      the array is dimensioned
      double[number_of_range_components][number_of_range_samples];
      copy array if copy flag is true */
  public void setSamples(double[][] range, boolean copy)
         throws VisADException, RemoteException;

  /** set range array as range values of this FlatField;
      the array is dimensioned
      double[number_of_range_components][number_of_range_samples];
      copy array if copy flag is true */
  public void setSamples(float[][] range, boolean copy)
         throws VisADException, RemoteException;

  /** get this FlatField's range values in their default range
      Units (as defined by the range of the FlatField's
      FunctionType); the return array is dimensioned
      double[number_of_range_components][number_of_range_samples] */
  public double[][] getValues()
         throws VisADException, RemoteException;
```

3.10 Immutable Data

Most Data classes and metadata classes are immutable, in order to ensure the thread-safeness of VisAD applications in distributed computing environments. The only exceptions

are Field and its sub-classes. Field metadata cannot change, but the values of Field and FlatField range samples can change (as well as the ErrorEstimates associated with FlatField range samples). Fields are mutable since they may be very large and it would be inefficient to have to copy them to change individual range values.

3.11 DataReferences

Since the only way to change the value of an immutable Data object is to replace it with a different Data object, there is a need for a class to represent variable Data. Thus the DataReference class defines mutable references to Data objects. In an application, for example, the variable *current_time* may be represented by a DataReference object that refers to a succession of immutable Real objects.

3.11.1 DataReference Constructors

DataReference is an interface that may apply to both local and remote DataReference objects. The DataReferenceImpl class applies only to local DataReference objects, while the RemoteDataReference interface and RemoteDataReferenceImpl class apply only to remote DataReference objects (see Section 6 for more information). The DataReference classes include the following constructors:

```
/** construct a DataReferenceImpl object with the given name */
public DataReferenceImpl(String name) throws VisADException;

/** construct a RemoteDataReferenceImpl object to provide remote
    access to reference */
public RemoteDataReferenceImpl(DataReferenceImpl reference)
       throws RemoteException;
```

3.11.2 DataReference Methods

Generally useful DataReference methods include:

```
/** get MathType of referenced Data object, or null if none;
    this is more efficient than getData().getType() for
    RemoteDataReferences */
public MathType getType() throws VisADException, RemoteException;

/** get referenced Data object, or null if none */
public Data getData() throws VisADException, RemoteException;

/** set reference to data, replacing any currently referenced
    Data object; if this is local (i.e., an instance of
    DataReferenceImpl) then the data argument must also be
    local (i.e., an instance of DataImpl);
    if this is Remote (i.e., an instance of RemoteDataReference)
    then a local data argument (i.e., an instance of DataImpl)
    will be passed by copy and a remote data argument (i.e., an
    instance of RemoteData) will be passed by remote reference */
public void setData(Data data) throws VisADException, RemoteException;
```

3.12 Application Example: Arrays versus VisAD Functions

In order to understand how to write numerical applications with VisAD, it is useful to compare VisAD with C. VisAD and C both allow applications to define complex data

structures from basic primitives. For example, a multi-spectral image can be defined in C using a structure and an array:

```
struct pixel {
  float ir_radiance;
  float vis_radiance;
};
struct pixel image[nlines][nelements];
```

A similar multi-spectral image can be defined in VisAD using RealTupleTypes and a FunctionType:

```
RealTupleType location = new RealTupleType(new RealType("line"),
                                           new RealType("element")),
RealTupleType pixel = new RealTupleType(new RealType("ir_radiance"),
                                        new RealType("vis_radiance")),
FunctionType image_type = new FunctionType(location, pixel);
Set location_set = new Integer2DSet(nlines, nelements);
FlatField image = new FlatField(image_type, location_set);
```

In general, we can list the following analogies between C and VisAD data structuring tools:

C	VisAD
float, double, int	RealType
char string[]	TextType
struct	TupleType, RealTupleType
array	FunctionType

In these analogies, C and VisAD syntax differ considerably. However, that kind of difference should be familiar to programmers with experience in several programming languages. The important similarities and differences relate to the meanings of these data structuring tools. Most differences involve metadata integrated into the meanings of data. For example, VisAD Reals and C floats implement the same set of operations, but operations on VisAD Reals may invoke Unit conversions and propogate ErrorEstimates and missing data indicators (some C implementations also propogate missing data indicators in the form of IEEE NaNs). C structs and VisAD Tuples have very similar meanings - they are both fixed length lists of other data structures. However, VisAD RealTuples may include CoordinateSystems and operations on RealTuples may invoke coordinate transforms.

The most complex differences exist for the analogy between C arrays and VisAD Functions, because of the variety of metadata integrated into VisAD Functions. The rest of this section of the Developers Guide is dedicated to explaining the relation between arrays and Functions in a series of program examples. With the proper understanding, you can use Functions anywhere you can use arrays, but Functions also allow you to express some very complex operations simply.

3.12.1 Subtracting Images as Pixel Arrays in C

The following C code could be used to compute the difference between two multi-spectral images:

```
#define nlines 256
#define nelements 256

struct pixel {
  float ir_radiance;
  float vis_radiance;
};

image_difference(image1, image2)
  struct pixel image1[nlines][nelements];
  struct pixel image2[nlines][nelements];
{
  int i, j;
  for (i=0; i<nlines; i++) {
    for (j=0; j<nelements; j++) {
      image1[i][j].ir_radiance -= image2[i][j].ir_radiance;
      image1[i][j].vis_radiance -= image2[i][j].vis_radiance;
    }
  }
}
```

This code assumes a fixed size for its image arguments, but that would not be hard to generalize. It also assumes a fixed set of spectral bands for its image arguments, that both images have the same size, that their pixel locations are aligned, and that image radiance values have the same units and calibration.

3.12.2 Subtracting Images as Pixel Arrays in VisAD

The following Java / VisAD code could be used to compute the difference between two multi-spectral images, in a pixel-by-pixel manner similar to the C code in Section 3.12.1:

```
void image_difference(FlatField image1, FlatField image2)
     throws VisADException, RemoteException {
  // extract pixel radiance values from images
  double[][] pixels1 = image1.getValues();
  double[][] pixels2 = image2.getValues();
  // loop over spectral bands in image1
  for (int i=0; i<pixels1.length; i++) {
    // loop over pixels in one spectral band
    for(int j=0; j<pixels1[i].length; j++) {
      pixels1[i][j] -= pixels2[i][j];
    }
  }
  // set pixel radiance values in image1
  image1.setSamples(pixels1);
}
```

This code does not assume a fixed size for its image arguments, and does not assume that they have only two spectral bands. However, it does assume that both images have the same size and the same set of spectral bands, that their pixel locations are aligned, and that image radiance values have the same units and calibration.

This code example demonstrates that it is easy to treat VisAD Functions like simple arrays, extracting their values into ordinary arrays using the *getValues* method and setting values from ordinary arrays using the *setSamples* method.

3.12.3 Subtracting Images as Functions in VisAD

The following Java / VisAD code computes the difference between two multi-spectral images at a high level, which allows VisAD to integrate all their metadata into the operation:

```
FlatField image_difference(FlatField image1, FlatField image2)
          throws VisADException, RemoteException {
  return (FlatField) image1.subtract(image2);
}
```

This code only assums that the two images have the same set of spectral bands. If necessary it will resample the locations of image2 to the locations of image1, transform locations from one coordinate system to another and convert location units, convert radiance units and transform between radiance calibration coordinate systems, and propogate error estimates and missing data indicators.

This code example demonstrates that Functions can be manipulated at a high level, similar to array operations in some high-level languages (such as IDL) but integrating a variety of metadata in those operations. High-level operations on Functions include basic arithmetic such as add and multiply with other Functions or with Reals, as well as derivative, resampling, and display.

4. Visualization

The basic visualization approach of VisAD can be summarized as:

1. Any number of interactive 3-D and 2-D displays can be created, each defined by a Display object. For example, Displays could be attached to each cell in a spread-sheet.
2. Each Display includes a set of ScalarMap objects that determine how Data objects are depicted. They define mappings from RealTypes (every primitive value occurring in a Data object has a RealType) to DisplayRealTypes (see Section 4.1).
3. Each Display includes links to any number of DataReference objects, depicted in a common frame of reference defined by the Display's ScalarMaps. Data depictions are updated whenever Data values change. In some cases, users can change Data values by re-drawing their depictions.

VisAD is designed to use a variety of graphic API's for generating Data displays. The current release of VisAD uses Java3D and Java2D. Java3D supports a wide variety of 3-D graphics techniques, while Java2D is part of the Java 1.2 core.

VisAD shields applications developers from details of graphics APIs. Thus for most applications, the only difference between Java3D and Java2D displays is whether they are constructed with DisplayImplJ3D or DisplayImplJ2D, and the constraint that Java2D displays cannot involve ScalarMaps to ZAxis, Latitude or Alpha.

The following description of the VisAD display architecture is complex but ordinary applications can use it quite simply, as illustrated by the application source code examples.

4.1 ScalarMaps and DisplayRealTypes

The simplest and most common way (see any issue of Science or Nature) to visualize numerical data is a 2-D plot of one physical quantity versus another, such as temperature versus pressure or humidity versus time. Scalar mappings generalize this idea to visualizations that are 3-D, animated, interactive, colored, transparent, etc. Every numerical value occurring in a Data object has a named RealType. ScalarMap objects define mappings from ScalarTypes to DisplayRealTypes, which are defined for all the primitive quantities of displays. The system defines a set of intrinsic DisplayRealTypes, and a set of groupings of these into DisplayTupleTypes, as public static final variables in the Display interface (so, for example, XAxis is accessed as Display.XAxis). The system-intrinsic DisplayRealTypes and DisplayTupleTypes are:

```
(XAxis, YAxis, ZAxis)          = DisplaySpatialCartesianTuple
(Latitude, Longitude, Radius) = DisplaySpatialSphericalTuple
(CylRadius, CylAzimuth, CylZAxis) = DisplaySpatialCylindricalTuple
(Red, Green, Blue)             = DisplayRGBTuple
(Hue, Saturation, Brightness) = DisplayHSBTuple
(Cyan, Magenta, Yellow)        = DisplayCMYTuple
RGB, HSV, CMY                  // indices into pseudo color table
RGBA                           // index into pseudo color-alpha table
Alpha                          // transparency
Animation                      // index into animation sequence
SelectValue, SelectRange       // select Data components for display
IsoContour                     // iso-contour lines and surfaces
(Flow1X, Flow1Y, Flow1Z)       = DisplayFlow1Tuple // vector rendering
(Flow2X, Flow2Y, Flow2Z)       = DisplayFlow2Tuple // 2nd vector set
(XAxisOffset, YAxisOffset, ZAxisOffset)  = DisplaySpatialOffsetTuple
Shape                          // index into list of icon shapes
ShapeScale                     // relative scale of icon shapes
Text                           // TextTypes only can be mapped to Text
LineWidth, PointSize           // for ConstantMap only
```

Developers may define new DisplayRealTypes and DisplayTupleTypes to define parameters of new kinds of displays, as described in Section 4.1.1. In particular developers may define new display spatial and color coordinate systems that can be used by existing DataRenderers and DisplayRenderers.

Some DisplayRealTypes define a range of values (e.g., 0.0 to 1.0). Values for these DisplayRealTypes are derived from mapped RealType values by linear scaling. The scale and offset are computed so that the range of RealType values is mapped precisely to the range of DisplayRealType values. Application can define the range of RealType values using the *setRange* method of ScalarMap, otherwise they are automatically computed from the displayed Data objects.

Each DisplayRealType defines a default value (most are 0.0, but for example the default for Radius is 1.0), which is over-ridden by values of any RealTypes mapped to the DisplayRealType. ConstantMap is a sub-class of ScalarMap and defines a mapping from a constant to a DisplayRealType (for example, to over-ride the default value for Radius). Each DataReference linked to a Display may also include its own private set of

ConstantMaps. This can be used, for example, to set a different color for each Data object, or to set a different ZAxis depth for each of a set of image Data objects displayed with transparent color in the XY plane.

The meanings of most DisplayRealTypes should be fairly obvious, but a few need some explanation. SelectRange and SelectValue are used to display only selected parts of Data objects, depending on whether values of RealTypes mapped to SelectRange lie in a specified range and whether values of RealTypes mapped to SelectValue have a specified value (this is only applicable to RealTypes occurring as 1-D Field domains and the value tolerance is defined according to the Field domain sampling Set). Animation is also only applicable to RealTypes occurring as 1-D Field domains and the discrete animation steps are defined from Field domain sampling Sets. The components of DisplaySpatialOffsetTuple are used to generate display spatial coordinates as the sums of values from multiple RealTypes. This could be used, for example, to define Beshers and Feiner's "worlds within worlds" display [2].

Display.Shape can be a very powerful tool for creating complex displays but is also a bit complex. See Section 4.7 for more information.

TextTypes may only be mapped to Display.Text, or left unmapped. RealTypes may not be mapped to Display.Text.

In Figure 1 (which is supplied with some hard copies of this guide, and is also available at http://www.ssec.wisc.edu/~billh/figure1.gif), the top-left panel shows a Data object with MathType:

```
( (nl, nchan) -> wfn )
```

displayed according to the mappings:

```
nl    -> YAxis        wfn -> ZAxis        0.5 -> Blue
nchan -> XAxis        wfn -> Green        0.5 -> Red
```

It is possible to map data to displays via the Reference RealTupleTypes of CoordinateSystems occurring in Data objects. For example, given a Data object with MathType:

```
( (lon, radius) -> (vis_radiance, ir_radiance) )
```

where (lon, radius) has a PolarCoordinateSystem with Reference (x, y), it is possible to display this Data object using the mappings:

```
x   -> XAxis        0.5 -> Blue       vis_radiance -> Green
y   -> YAxis        0.5 -> Red
```

This display can be seen with the command 'java DisplayTest 11' in the visad/exmaples directory.

Note that the *main* method of DisplayTest provides many examples of how ScalarMaps can be used.

Classes that implement the ScalarMapListener interface can be attached to ScalarMaps via their *addScalarMapListener* method. ScalarMaps send ScalarMapEvents

to attached ScalarMapListeners when their range of values is changed by display autoscaling. ScalarMapEvents include a reference to the ScalarMap that generated them, and ScalarMaps are Serializable, so they can be used between different JVMs (i.e., different computers or different Java interpreters on the same computer).

4.1.1 Common Sense and ScalarMaps

Not all mappings from RealTypes to DisplayRealTypes are legal, and legality may depend on the MathTypes of Data objects linked to the Display. The constraints on ScalarMaps and MathTypes used by the DefaultDisplayRendererJ3D and DefaultDataRendererJ3D classes are described in Appendix A. Most intuitive combinations are legal. Illegal combinations result in BadMappingExceptions. Legal combinations that are not yet implemented result in UnimplementedExceptions. These Exceptions are displayed at the bottom of the display window.

Rather than focusing on the complex constraints described in Appendix A it is easiest to apply common sense in defining ScalarMaps.

The RealType components of FlatField domains should be mapped to XAxis, YAxis and ZAxis or to Latitude, Longitude and Radius (but note that Cartesian and spherical spatial coordinates cannot be mixed). The RealType components of FlatField ranges should be mapped to one of the color DisplayRealTypes (e.g., Green, RGB), Alpha (transparency), IsoContour, one of the flow DisplayRealTypes (e.g., Flow1X, Flow1Y), Shape (although note that Shape is not implemented in the initial release of VisAD) or to a spatial coordinate not mapped from the domain (note that multiple RealType components from the same FlatField cannot be mapped to the same spatial coordinates).

However, in order to produce scatter plots of FlatField range values (e.g., scatter plots relating the different radiance channels of a satellite image) the RealType components of the FlatField domain should generally not be mapped while the RealType components of the range (i.e., the different radiance channel types) should be mapped to spatial coordinates and to color DisplayRealTypes (for colored scatter plots).

When FunctionTypes are nested in the ranges of other FunctionTypes (for example, a time sequence of images) RealType components of the outer Field domain should be mapped to Animation, SelectValue, SelectRange, color DisplayRealTypes, and spatial offsets (e.g., XAxisOffset). However, note that only RealType components of 1-D Field domains may be mapped to Animation or SelectValue.

When Tuples include RealType components and FunctionType components, DisplayRealTypes mapped from the RealType components will affect the depiction of the FunctionType components. They should be mapped to color DisplayRealTypes, spatial offsets and SelectRange.

4.1.2 DisplayRealType and DisplayTupleType Constructors

Developers may define parameters of new kinds of displays using the DisplayRealType and DisplayTupleType constructors. Generally new DisplayRealTypes and DisplayTupleTypes will require developers to extend DataRenderer and possible DisplayRenderer. However, the current Java3D and Java2D DataRenderers can handle new DisplayRealTypes that are components of new DisplayTupleTypes whose CoordinateSystems have Reference that is either DisplaySpatialCartesianTuple or

DisplayRGBTuple. In these cases the developer is defining new display spatial coordinate systems and new display color coordinate systems. The constructors are:

```
/** construct a DisplayRealType with given name (used only for
    user interfaces), single flag (if true, this DisplayRealType
    may only occur once in a path to a terminal node, as defined
    in Appendix A), (low, hi) range of values, default value,
    and unit */
public DisplayRealType(String name, boolean single, double low,
                       double hi, double default, Unit unit)
       throws VisADException;

/** similar to above constructor but without value range;
    values of RealTypes mapped to this DisplayRealType are
    not scaled */
public DisplayRealType(String name, boolean single,
                       double default, Unit unit)
       throws VisADException;

/** if coord_sys is not null then coord_sys.Reference
    must be another DisplayTupleType; a DisplayrealType may
    not be a component of more than one DisplayTupleType */
public DisplayTupleType(DisplayRealType[] types,
                        CoordinateSystem coord_sys)
       throws VisADException;

public DisplayTupleType(DisplayRealType[] types)
       throws VisADException;
```

4.1.3 DisplayRealType Methods Useful for Extending DataRenderer

The methods of DisplayRealType are only useful to developers who extend DataRenderer. They include:

```
/** return the unique DisplayTupleType that this
    DisplayRealType is a component of, or return null
    if it is not a component of any DisplayTupleType */
public DisplayTupleType getTuple();

/** return index of this as component of a
    DisplayTupleType */
public getTupleIndex();

/** return true if this DisplayRealType is 'single' */
public isSingle();

/** return default value for this DisplayRealType */
public double getDefaultValue();

/** return true is a range of values is defined for
    this DisplayRealType, and return the range in
    range[0] and range[1]; range must be passed in
    as a double[2] array */
public boolean getRange(double[] range);
```

4.1.4 ScalarMap and ConstantMap Constructors

The ScalarMap class and its ConstantMap subclass are serializable. ScalarMap objects may only be local. The ScalarMap class include the following constructors:

```
public ScalarMap(ScalarType scalar, DisplayRealType display_scalar)
       throws VisADException;

/** construct a ConstantMap with a double constant;
    display_scalar may not be Animation, SelectValue, SelectRange
    or IsoContour */
public ContantMap(double constant, DisplayRealType display_scalar)
       throws VisADException;

/** construct a ConstantMap with a Real constant;
    display_scalar may not be Animation, SelectValue, SelectRange
    or IsoContour */
public ContantMap(Real constant, DisplayRealType display_scalar)
       throws VisADException;
```

4.1.5 Generally Useful ScalarMap Methods

Generally useful ScalarMap methods include:

```
public ScalarType getScalar();

public DisplayRealType getDisplayScalar();

/** get the Control this ScalarMap is linked to;
    the Control is constructed when this ScalarMap is linked to
    a Display via an invocation of the Display's addMap method;
    not all ScalarMaps have Controls, generally depending on the
    ScalarMap's DisplayRealType */
public Control getControl();

/** return value is true if data (RealType) values are linearly
    scaled to display (DisplayRealType) values;
    if so, then values are scaled by:
    display_value = data_value * scale_offset[0] + scale_offset[1];
    (data[0], data[1]) defines range of data values (either passed
    in to setRange or computed by autoscaling logic) and
    (display[0], display[1]) defines range of display values;
    scale_offset, data, display must each be passed in as
    double[2] arrays */
public boolean getScale(double[] scale_offset, double[] data,
                        double[] display);

/** explicitly set the range of data (RealType) values;
    if neither this nor setRangeByUnits is invoked, then the
    range will be computed from the initial values of Data
    objects linked to the Display by autoscaling logic;
    if the range of data values is (0.0, 1.0), for example, this
    method may be invoked with low = 1.0 and hi = 0.0 to invert
    the display scale */
public void setRange(double low, double hi)
       throws VisADException, RemoteException;

/** explicitly set the range of data (RealType) values according
    to Unit conversion between this ScalarMap's RealType and
    DisplayRealType (both must have Units and they must be
```

```
    convertable; if neither this nor setRange is invoked, then
    the range will be computed from the initial values of Data
    objects linked to the Display by autoscaling logic */
public void setRangeByUnits() throws VisADException, RemoteException;

/** set color of axis scales; color must be float[3] with red,
    green and blue components; DisplayScalar must be XAxis,
    YAxis or ZAxis */
public void setScaleColor(float[] color)
       throws VisADException;

/** add a ScalarMapListener */
public void addScalarMapListener(ScalarMapListener listener);

/** remove a ScalarMapListener */
public void removeScalarMapListener(ScalarMapListener listener);
```

4.1.6 ScalarMap Methods Useful for Extending DataRenderer

Some ScalarMap methods are useful only for extending the DataRenderer class. These include:

```
/** return an array of display (DisplayRealType) values by
    linear scaling (if applicable) the data_values array
    (RealType values) */
public float[] scaleValues(double[] data_values);

public float[] scaleValues(float[] data_values);

/** return an array of data (RealType) values by inverse
    linear scaling (if applicable) the display_values array
    (DisplayRealType values); this is useful for direct
    manipulation and cursor labels */
public float[] inverseScaleValues(float[] display_values);
```

4.1.7 ConstantMap Methods

Although ConstantMap extends ScalarMap, most ScalarMap methods do not make sense for ConstantMaps, except for *getDisplayScalar*. Generally useful ConstantMap methods include:

```
public double getConstant();
```

4.1.8 ScalarMapListener Methods

ScalarMapListener is an interface that extends EventListener.

```
/** send a ScalarMapEvent to this ScalarMapListener */
public void mapChanged(ScalarMapEvent event)
       throws VisADException, RemoteException;
```

4.1.9 ScalarMapEvent Methods

ScalarMapEvent is a class that extends Event.

```
/** get the ScalarMap that sent this ScalarMapEvent (or
    a copy if the ScalarMap was on a different JVM) */
public ScalarMap getScalarMap();
```

4.1.10 Application Example: ScalarMaps and ConstantMaps

Assume a Data object named 'images' that is a time sequence of multi-spectral images with MathType:

```
(time -> ((line, element) -> (ir_radiance, vis_radiance)))
```

The following code could be used to generate four different displays of the 'images' Data object:

```
// generate a traditional image display with ir radiances mapped
// to red, visible radiances mapped to green, constant blue,
// and animating over the time sequence;
// NOTE - this display can take adavntage of texture mapping
// for efficiency
display1 = new DisplayImplJ3D("display1");
display1.addMap(new ScalarMap(time, Display.Animation));
display1.addMap(new ScalarMap(line, Display.YAxis));
display1.addMap(new ScalarMap(element, Display.XAxis));
display1.addMap(new ScalarMap(ir_radiance, Display.Red));
display1.addMap(new ScalarMap(vis_radiance, Display.Green));
display1.addMap(new ConstantMap(0.5, Display.Blue));

// visualize the images as contour lines of visible radiance
// on a 3-D terrain surface defined by ir radiances, with the
// contours colored by visible radiances and animating over
// the time sequence
display2 = new DisplayImplJ3D("display2");
display2.addMap(new ScalarMap(time, Display.Animation));
display2.addMap(new ScalarMap(line, Display.YAxis));
display2.addMap(new ScalarMap(element, Display.XAxis));
display2.addMap(new ScalarMap(ir_radiance, Display.ZAxis));
display2.addMap(new ScalarMap(vis_radiance, Display.IsoContour));
display2.addMap(new ScalarMap(vis_radiance, Display.RGB));

// visualize the images as 2-D scatter diagrams of ir
// radiance versus visible radiance, with points colored by
// time
display3 = new DisplayImplJ3D("display3");
display3.addMap(new ScalarMap(ir_radiance, Display.XAxis));
display3.addMap(new ScalarMap(vis_radiance, Display.YAxis));
display3.addMap(new ScalarMap(time, Display.RGB));

// generate a set of traditional image displays (i.e.,
// similar to display1) but with the time sequence stacked
// up in the vertical (ZAxis) rather than animated
display4 = new DisplayImplJ3D("display4");
display4.addMap(new ScalarMap(time, Display.ZAxis));
display4.addMap(new ScalarMap(line, Display.YAxis));
display4.addMap(new ScalarMap(element, Display.XAxis));
display4.addMap(new ScalarMap(ir_radiance, Display.Red));
display4.addMap(new ScalarMap(vis_radiance, Display.Green));
display4.addMap(new ConstantMap(0.5, Display.Blue));
```

4.2 DataRenderers and DisplayRenderers

Data display is a two step process:

1. Data objects are transformed into graphical display lists (e.g., Java3D scene graphs). This is done by objects of the DataRenderer and DisplayRenderer class hierarchies.
2. Display lists are rendered.

A Display has one DisplayRenderer object: it manages the display lists produced for all Data linked to the Display, it manages mouse events in the Display window and their connection to Controls (e.g., rotating the 3-D scene by dragging the mouse), it renders display axes, cursors, labels and error messages, and it adds any specialized metadata rendering (e.g., the background wet and dry adiabats in a skew-t diagram). A Display may have several DataRenderer objects, each linked to one or more of the Display's DataReference objects. Each DataRenderer transforms its set of referenced Data objects into a display list, and is responsible for the consistency of that transformation with the Display's ScalarMaps. Developers may ignore the issue of DataRenderers by using the *addReference* method of Display rather than the *addReferences* method, in which case Displays use their default DisplayRenderers (and each DataReference is linked to a different instance of a default DataRenderer).

Developers have the option to extend the DataRenderer and DisplayRenderer classes in order to customize Data displays. In fact, developers will need to extend the DataRenderer and DisplayRenderer classes for most extensions of the DisplayRealType and DisplayTupleType classes, because existing DataRenderer and DisplayRenderer classes will not know what to do with developer-defined DisplayRealType and DisplayTupleType classes (unless they are related to existing DisplayRealType and DisplayTupleType classes via CoordinateSystem References).

4.2.1 Java3D DataRenderer and DisplayRenderer Constructors

DataRenderer and DisplayRenderer are abstract classes whose concrete subclasses are specific to particular graphics APIs. The visad.java3d package defines classes specific to the Java3D graphics API. The Java3D DataRenderer and DisplayRenderer constructors include:

```
/** this is the default DataRenderer used by the addReference method
    for DisplayImplJ3D */
public DefaultRendererJ3D();

/** this DataRenderer supports direct manipulation for Real,
    RealTuple and Field Data objects (Field data objects must
    have RealType or RealTupleType ranges and Gridded1DSet
    domain Sets); no RealType may be mapped to multiple spatial
    DisplayRealTypes; the RealType of a Real object must be
    mapped to XAxis, YAxis or YAxis; at least one of the
    RealType components of a RealTuple object must be mapped
    to XAxis, YAxis or YAxis; the domain RealType and at
    least one RealType range component of a Field object
    must be mapped to XAxis, YAxis or YAxis */
public DirectManipulationRendererJ3D();

/** this is the default DisplayRenderer used by the
    DisplayImplJ3D constructor;
    it draws a 3-D cube around the scene;
```

```
    the left mouse button controls the projection as
    follows: mouse drag rotates in 3-D, mouse drag with
    Shift down zooms the scene, mouse drag with Ctrl
    translates the scene sideways;
    the center mouse button activates and controls the
    3-D cursor as follows: mouse drag translates the
    cursor sideways, mouse drag with Shift translates
    the cursor in and out, mouse drag with Ctrl rotates
    scene in 3-D with cursor on;
    the right mouse button is used for direct
    manipulation by clicking on the depiction of a Data
    object and dragging or re-drawing it;
    cursor and direct manipulation locations are displayed
    in RealType values;
    BadMappingExceptions and UnimplementedExceptions are
    displayed */
  public DefaultDisplayRendererJ3D();

  /** this DisplayRenderer supports 2-D only rendering;
      is easiest to describe in terms of differences
      from DefaultDisplayRendererJ3D: the cursor and box
      around the scene are 2-D, the scene cannot be rotated,
      the cursor cannot be translated in and out, and the
      scene can be translated sideways with the left mouse
      button with or without pressing the Ctrl key;
      no RealType may be mapped to ZAxis or Latitude */
  public TwoDDisplayRendererJ3D();
```

4.2.2 Java2D DataRenderer and DisplayRenderer Constructors

DataRenderer and DisplayRenderer are abstract classes whose concrete subclasses are specific to particular graphics APIs. The visad.java2d package defines classes specific to the Java2D graphics API. The Java2D DataRenderer and DisplayRenderer constructors include:

```
  /** this is the default DataRenderer used by the addReference method
      for DisplayImplJ2D */
  public DefaultRendererJ2D();

  /** this DataRenderer supports direct manipulation for Real,
      RealTuple and Field Data objects (Field data objects must
      have RealType or RealTupleType ranges and Gridded1DSet
      domain Sets); no RealType may be mapped to multiple spatial
      DisplayRealTypes; the RealType of a Real object must be
      mapped to XAxis, YAxis or YAxis; at least one of the
      RealType components of a RealTuple object must be mapped
      to XAxis, YAxis or YAxis; the domain RealType and at
      least one RealType range component of a Field object
      must be mapped to XAxis, YAxis or YAxis */
  public DirectManipulationRendererJ2D();

  /** this is the default DisplayRenderer used by the
      DisplayImplJ2D constructor;
      it draws a 2-D box around the scene and a 2-D cursor;
      the left mouse button controls the projection as
      follows: mouse drag or mouse drag with Ctrl translates
      the scene sideways, mouse drag with Shift down zooms
      the scene; the center mouse button activates and
```

```
    controls the 2-D cursor as follows: mouse drag
    translates the cursor sideways; the right mouse button
    is used for direct manipulation by clicking on the
    depiction of a Data object and dragging or re-drawing
    it; cursor and direct manipulation locations are
    displayed in RealType values; BadMappingExceptions
    and UnimplementedExceptions are displayed;
    no RealType may be mapped to ZAxis, Latitude
    or Alpha */
public DefaultDisplayRendererJ2D();
```

4.2.3 DataRenderer Methods

Developers who extend the DataRenderer and DisplayRenderer classes should be aware of the following DataRenderer methods:

```
/** this returns a Vector of Strings from the BadMappingExceptions
    and UnimplementedExceptions generated during the last invocation
    of this DataRenderer's doAction method;
    there is no need to over-ride this method, but it may be invoked
    by DisplayRenderer */
public Vector getExceptionVector();

/** return an array of links to Data objects to be rendered;
    Data objects are accessed by DataDisplayLink.getData() */
public DataDisplayLink[] getLinks();

/** transform linked Data objects into a display list, if
    any Data object values have changed or relevant Controls
    have changed; DataRenderers that assume the default
    implementation of DisplayImpl.doAction can determine
    whether re-transform is needed by:
      (all_feasible && (any_changed || any_transform_control));
    these flags are computed by the default DataRenderer
    implementation of prepareAction;
    the return boolean is true if the transform was done
    successfully */
public abstract boolean doAction()
       throws VisADException, RemoteException;

/** set isDirectManipulation = true if this DataRenderer
    supports direct manipulation for its linked Data */
public void checkDirect()
       throws VisADException, RemoteException;

/** clear any display list created by the most recent doAction
    invocation */
public abstract void clearScene();

/** factory for constructing a subclass of ShadowType appropriate
    for the graphics API, that also adapts ShadowFunctionType;
    these factories are invoked by the buildShadowType methods of
    the MathType subclasses, which are invoked by
    DataDisplayLink.prepareData, which is invoked by
    DataRenderer.prepareAction */
public abstract ShadowType makeShadowFunctionType(
       FunctionType type, DataDisplayLink link, ShadowType parent)
       throws VisADException, RemoteException;
```

```
  /** factory for constructing a subclass of ShadowType appropriate
      for the graphics API, that also adapts ShadowRealTupleType */
  public abstract ShadowType makeShadowRealTupleType(
         RealTupleType type, DataDisplayLink link, ShadowType parent)
         throws VisADException, RemoteException;

  /** factory for constructing a subclass of ShadowType appropriate
      for the graphics API, that also adapts ShadowRealType */
  public abstract ShadowType makeShadowRealType(
         RealType type, DataDisplayLink link, ShadowType parent)
         throws VisADException, RemoteException;

  /** factory for constructing a subclass of ShadowType appropriate
      for the graphics API, that also adapts ShadowSetType */
  public abstract ShadowType makeShadowSetType(
         SetType type, DataDisplayLink link, ShadowType parent)
         throws VisADException, RemoteException;

  /** factory for constructing a subclass of ShadowType appropriate
      for the graphics API, that also adapts ShadowTextType */
  public abstract ShadowType makeShadowTextType(
         TextType type, DataDisplayLink link, ShadowType parent)
         throws VisADException, RemoteException;

  /** factory for constructing a subclass of ShadowType appropriate
      for the graphics API, that also adapts ShadowTupleType */
  public abstract ShadowType makeShadowTupleType(
         TupleType type, DataDisplayLink link, ShadowType parent)
         throws VisADException, RemoteException;

  /** return true if a change in control requires re-transform;
      this decision may use some values computed by
      link.prepareData */
  public boolean isTransformControl(Control control,
                                    DataDisplayLink link);
```

4.2.4 DisplayRenderer Methods

The *setBoxOn*, *setBoxColor*, *setCursorColor* and *setBackgroundColor* methods are of general use. The other methods in this section are useful to developers who extend the DataRenderer and DisplayRenderer classes.

```
  /** set display box on or off */
  public void setBoxOn(boolean on);

  /** set color of display box */
  public void setBoxColor(float r, float g, float b);

  /** set color of display cursor */
  public void setCursorColor(float r, float g, float b);

  /** set color of display window background */
  public void setBackgroundColor(float r, float g, float b);

  /** return the DisplayImpl that this DisplayRenderer is attached to */
  public DisplayImpl getDisplay();

  /** return true is this is a 2-D DisplayRenderer */
  public boolean getMode2D();
```

```
/** factory for constructing a subclass of Control appropriate
    for the graphics API and for this DisplayRenderer;
    invoked by ScalarMap when it is added to a Display */
public abstract Control makeControl(DisplayRealType type);

/** factory for constructing the default subclass of
    DataRenderer for this DisplayRenderer */
public abstract DataRenderer makeDefaultRenderer();

/** return a double[3] array giving the cursor location in
    (XAxis, YAxis, ZAxis) coordinates */
public double[] getCursor();

/** return Vector of Strings describing the cursor location */
public Vector getCursorStringVector();

/** set vector of Strings describing the cursor location
    from the cursor location;
    this is invoked when the cursor location changes or
    the cursor display status changes */
public void setCursorStringVector();

/** set vector of Strings describing the cursor location;
    this is invoked by direct manipulation renderers */
public void setCursorStringVector(Vector vector);

/** return true if type is legal for this DisplayRenderer;
    for example, 2-D DisplayRenderers use this to disallow
    mappings to ZAxis and Latitude */
public boolean legalDisplayScalar(DisplayRealType type);
```

4.2.5 DisplayRendererJ2D Method

The following method is only implemented for Java2D.

```
/** set clipping bounds */
public void setClip(float xlow, float xhi, float ylow, float yhi);
```

4.3 Controls

Because VisAD has no intrinsic user interface, the Control class hierarchy takes the place of visualization user interface components. The class hierarchy is:

```
Control
  AnimationControl    // animation stepping (interface)
  AnimationSetControl // animation sampling
  ColorControl        // pseudo color table (for RGB, CMY, HSV, etc)
  ColorAlphaControl   // pseudo color-alpha table (for RGBA, etc)
  ContourControl      // iso-contour levels and intervals
  FlowControl         // flow rendering
    Flow1Control      // render 1st set of flow vectors
    Flow2Control      // render 2nd set of flow vectors
  GraphicsModeControl // line width, point size, etc (interface)
  ProjectionControl   // 3-D rotation, scaling, translation (interface)
  RangeControl        // ranges of values
  ShapeControl        // array of shapes
  ToggleControl       // toggle other Controls on and off
  ValueControl        // individual value (interface)
```

```
TextControl         // text plotting of Text values
```

Developers can extend the Control class to define new types of Controls for new DisplayRealTypes (the binding from DisplayRealType to Control is defined in the *makeControl* method of DisplayRenderer). Developers may also extend subclasses of Control to define new forms of interaction for existing DisplayRealTypes.

AnimationControl, GraphicsModeControl, ProjectionControl and ValueControl are interfaces rather than classes, which must be implemented in a graphics-API-dependent way.

Instances of Control are linked to instances of ScalarMap. For some Control sub-classes, such as ProjectionControl and GraphicsModeControl, only one instance exists per Display. For other Control sub-classes, such as ContourControl, one instance exists per linked ScalarMap. Note that GraphicsModeControl is not linked to any instance of ScalarMap and every Display has a ProjectionControl even if no RealTypes are mapped to display spatial coordinates.

State changes in Controls may trigger a re-transformation of affected Data objects via their DataRenderers, or may not. For example, changes in a ProjectionControl will not generally trigger re-transformation, while changes in a ContourControl will trigger re-transformation of Data whose component RealTypes are mapped to IsoContour via the associated ScalarMap. DisplayRenderers are responsible for building any links from 3-D graphics APIs to Controls (e.g., so that mouse movements trigger changes in ProjectionControl to rotate, zoom and translate the 3-D display).

Classes that implement the ControlListener interface can be attached to Controls via their *addControlListener* method. Controls send ControlEvents to attached ControlListeners whenever they change state. ControlEvents include a reference to the Control that generated them, and Controls are Serializable, so that Displays on different JVMs (i.e., different computers or different Java interpreters on the same computer) can exchange ControlEvents and Controls to implement collaborative visualization.

4.3.1 Control Methods

Control is an abstract class.

```
/** add a ControlListener */
public void addControlListener(ControlListener listener);

/** remove a ControlListener */
public void removeControlListener(ControlListener listener);
```

4.3.2 ControlListener Methods

ControlListener is an interface that extends EventListener.

```
/** send a ControlEvent to this ControlListener */
public void controlChanged(ControlEvent event)
       throws VisADException, RemoteException;
```

4.3.3 ControlEvent Methods

ControlEvent is a class that extends Event.

```
  /** get the Control that sent this ControlEvent (or a copy
      if the Control was on a different JVM) */
  public Control getControl();
```

4.3.4 AnimationControl Methods

Implementations of the AnimationControl interface are runnable in order to implement automatic animation stepping. Generally useful methods of AnimationControl include:

```
  /** set the current ordinal step number */
  public void setCurrent(int number)
         throws VisADException, RemoteException;

  /** set the current step by the value of the RealType
      mapped to Display.Animation */
  public void setCurrent(float value)
         throws VisADException, RemoteException;

  /** get the current ordinal step number */
  public int getCurrent();

  /** true for forward, false for backward */
  public void setDirection(boolean direction)
         throws VisADException, RemoteException;

  /** set the dwell time for each step, in milliseconds */
  public void setStep(int ms)
         throws VisADException, RemoteException;

  /** advance one step (forward or backward) */
  public void takeStep()
         throws VisADException, RemoteException;

  /** turn on automatic stepping if on = true, turn it
      off if on = false */
  public void setOn(boolean on)
         throws VisADException, RemoteException;

  /** return true if automatic stepping is on */
  public boolean getOn();

  /** toggle automatic stepping between off and on */
  public void toggle()
         throws VisADException, RemoteException;

  /** get Set of RealType values for animation steps */
  public Set getSet();
```

4.3.5 ColorControl Methods

Generally useful methods of ColorControl include:

```
  /** define the color lookup by a Function, whose MathType must
      have a 1-D domain and a 3-D RealTupleType range; the domain
      and range Reals must vary over the range (0.0, 1.0) */
  public void setFunction(Function function)
```

```
        throws VisADException, RemoteException;

  /** define the color lookup by an array of floats which must
      have the form float[3][table_length]; values should be in
      the range (0.0, 1.0) */
  public void setTable(float[][] table)
        throws VisADException, RemoteException;
```

4.3.6 ColorAlphaControl Methods

Generally useful methods of ColorAlphaControl include:

```
  /** define the color lookup by a Function, whose MathType must
      have a 1-D domain and a 4-D RealTupleType range; the domain
      and range Reals must vary over the range (0.0, 1.0) */
  public void setFunction(Function function)
        throws VisADException, RemoteException;

  /** define the color lookup by an array of floats which must
      have the form float[4][table_length]; values should be in
      the range (0.0, 1.0) */
  public void setTable(float[][] table)
        throws VisADException, RemoteException;
```

4.3.7 ContourControl Methods

Generally useful methods of ContourControl include:

```
  /** set level for iso-surfaces */
  public void setSurfaceValue(float value)
        throws VisADException, RemoteException;

  / ** set parameters for iso-lines: draw lines for levels
       between low and hi, starting at base, spaced by
       interval */
  public void setContourInterval(float interval, float low,
                                 float hi, float base)
        throws VisADException, RemoteException;

  / ** set array of unevenly spaced iso-line levels; if dash is
       true then iso-lines for levels below base are dashed */
  public void setLevels(float[] levels, float base, boolean dash)
        throws VisADException, RemoteException;

  /** enable contours */
  public void enableContours(boolean on)
        throws VisADException, RemoteException;

  /** enable labels */
  public void enableLabels(boolean on)
        throws VisADException, RemoteException;

  /** get contour parameters: bvalues[0] = contour enable,
      bvalues[1] = labels enable, fvalues[0] = surface level,
      fvalues[1] = interval, fvalues[2] = low, fvalues[3] = hi,
      fvalues[4] = base; bvalues and fvalues must be passed in
      as boolean[2] and float[5] */
  public void getMainContours(boolean[] bvalues, float[] fvalues)
        throws VisADException;
```

4.3.8 FlowControl Methods

Generally useful methods of FlowControl include:

```
/** set scale length for flow vectors (default is 0.02f) */
public void setFlowScale(float scale)
       throws VisADException, RemoteException;
```

4.3.9 GraphicsModeControl Methods

Generally useful methods of GraphicsModeControl include:

```
/** if enable is true this will enable numerical
    scales along display spatial axes; default is false */
public setScaleEnable(boolean enable)
       throws VisADException, RemoteException;

/** set the width of line rendering; this is over-ridden by
    ConstantMaps to Display.LineWidth; default is 1 */
public void setLineWidth(float width)
       throws VisADException, RemoteException;

/** set the size for point rendering; this is over-ridden by
    ConstantMaps to Display.PointSize; default is 1 */
public void setPointSize(float size)
       throws VisADException, RemoteException;

/** if mode is true this will cause some rendering as points
    rather than lines or surfaces; default is false */
public void setPointMode(boolean mode)
       throws VisADException, RemoteException;

/** if enable is true this will enable use of texture
    mapping, where appropriate; default is true */
public void setTextureEnable(boolean enable)
       throws VisADException, RemoteException;

/** sets a graphics-API-specific transparency mode (e.g.,
    SCREEN_DOOR, BLENDED); default is FASTEST */
public void setTransparencyMode(int mode)
       throws VisADException, RemoteException;

/** sets a graphics-API-specific projection policy (e.g.,
    PARALLEL_PROJECTION, PERSPECTIVE_PROJECTION);
    default is PERSPECTIVE_PROJECTION */
public void setProjectionPolicy(int policy)
       throws VisADException, RemoteException;

/** if transparent is true missing data are made transparent
    rather than just black; default is false */
public void setMissingTransparent(boolean transparent)
       throws VisADException, RemoteException;

/** size of flat areas in curved texture mapping; if size
    is less than 1 then curved texture maps are disabled;
    default size is 10 */
public void setCurvedSize(int size)
       throws VisADException, RemoteException;
```

4.3.10 ProjectionControl Methods

Generally useful methods of ProjectionControl include:

```
/** set the 4x4 matrix that defines the graphics
    projection */
public void setMatrix(double[] matrix)
       throws VisADException, RemoteException;

/** get the 4x4 matrix that defines the graphics
    projection */
public double[] getMatrix();
```

4.3.11 RangeControl Methods

Generally useful methods of RangeControl include:

```
/** set the range of selected values as (range[0], range[1]) */
public void setRange(float[] range)
       throws VisADException, RemoteException;

/** return the range of selected values */
public float[] getRange();
```

4.3.12 ShapeControl Methods

Generally useful methods of ShapeControl include:

```
/** set the SimpleSet that defines the mapping from RealType
    values to indices into an array of shapes;
    the domain dimension of set must be 1 */
public void setShapeSet(SimpleSet set)
       throws VisADException, RemoteException;

/** set the shape associated with index;
    the VisADGeometryArray class hierarchy defines various
    kinds of shapes */
public void setShape(int index, VisADGeometryArray shape)
       throws VisADException, RemoteException;

/** set the array of shapes associated with indices 0
    through shapes.length; the VisADGeometryArray class
    hierarchy defines various kinds of shapes */
public void setShapes(VisADGeometryArray[] shapes)
       throws VisADException, RemoteException;
```

4.3.13 ValueControl Methods

Generally useful methods of ValueControl include:

```
/** set the selected value */
public void setValue(float value)
       throws VisADException, RemoteException;

/** return the selected value */
public float getValue();
```

4.3.14 TextControl Methods

Generally useful methods of TextControl include:

```
/** set the size of characters; the default is 1.0 */
public void setSize(double size)
       throws VisADException, RemoteException;

/** return the size */
public double getSize();

/** set the centering flag; if true, text will be centered at
    mapped locations; if false, text will be to the right
    of mapped locations */
public void setCenter(boolean center)
       throws VisADException, RemoteException;

/** return the centering flag */
public boolean getCenter();

/** set the Font; in the initial release this has no effect */
public void setFont(Font font)
       throws VisADException, RemoteException;

/** return the Font */
public Font getFont();
```

4.4 Mouse Interactions and Direct Manipulation

Direct manipulation refers to user interface components embedded in the interactive 3-D display. This includes simple interactions, like rotating the scene in 3-D by dragging the mouse, and complex interactions like changing Data values by re-drawing their depictions.

The DefaultDisplayRendererJ3D class supports the following mouse interactions using Java3D:

1. The left mouse button controls the projection as follows: mouse drag rotates in 3-D, mouse drag with Shift down zooms the scene, mouse drag with Ctrl translates the scene sideways.
2. The center mouse button activates and controls the 3-D cursor as follows: mouse drag translates the cursor sideways, mouse drag with Shift translates the cursor in and out, mouse drag with Ctrl rotates scene in 3-D with cursor on.
3. The right mouse button is used for direct manipulation by clicking on the depiction of a Data object and dragging or re-drawing it.

Pressing any two buttons simulates pressing the third button, in order to accomodate two button mice.

3-D cursor and direct manipulation locations are displayed in the upper left corner of the display window as RealType values with Units. BadMappingExceptions and UnimplementedExceptions are displayed at the bottom of the display window. Animation information is displayed in the upper right corner of the display window as RealType values with Units.

The TwoDDisplayRendererJ3D and DefaultDisplayRendererJ2D classes support similar mouse interactions, with the following exceptions: the scene cannot be rotated, the 3-D cursor cannot be translated in and out, and the scene may translated sideways with the left mouse button without the need to press the Ctrl key.

4.4.1 Changing Data Values by Re-drawing Data Depictions

VisAD includes special extensions of the DataRenderer class, currently the DirectManipulationRendererJ3D class using Java3D and the DirectManipulationRendererJ2D class using Java2D, that allow users to modify Data objects by re-drawing their depictions. The DirectManipulationRendererJ3D and DirectManipulationRendererJ2D classes only support direct manipulation of Real, RealTuple and Field Data objects (Field data objects must have RealType or RealTupleType ranges and Gridded1DSet domain Sets). They also imposes the following restrictions on ScalarMaps:

1. At least one RealType in the Data object's MathType must be mapped to a spatial DisplayRealType.
2. No RealType may be mapped to multiple spatial DisplayRealTypes.
3. RealTypes may not be mapped to spatial DisplayRealTypes in multiple DisplayTupleTypes.
4. If the mapped spatial DisplayTupleType is not DisplaySpatialCartesianTuple, then RealTypes must be mapped to three spatial DisplayRealTypes (two if this is a Java2D display or a Java3D display with TwoDDisplayRendererJ3D).
5. The RealType of a Real object must be mapped to a spatial DisplayRealType.
6. At least one of the RealType components of a RealTuple object must be mapped to a spatial DisplayRealType.
7. The domain RealType and at least one RealType range component of a Field object must be mapped to a spatial DisplayRealType.

Data depictions are re-drawn by clicking the right mouse button while the mouse cursor is on the Data depiction. If the user has successfully picked a Data object, the coordinates of the selected Data point will be displayed in the upper left corner of the Display window. As the user drags the mouse Data values will change according to whatever degrees of freedom are possible according to the MathType and the ScalarMaps. In particular, direct manipulation can change range values of a Field but cannot change its domain Set.

Note that direct manipulation rendering using display spatial coordinate transforms can only be done when the mappings of the Data object's MathType spans the full spatial dimensionality of the display (3-D or 2-D). This restriction avoids the ambiguity of finding the closest point to a line on a curved submanifold. Test number 40 of DisplayTest illustrates direct manipulation linking Cartesian and polar display coordinates.

4.4.2 Application Example: Interactive Scaling

This is a section of code that illustrates how an application can build interactive scaling of Display spatial axes, through combined use of Display Controls, direct

manipulation, and computation Cells (described in Section 5). This is actually implemented by test number 27 of the DisplayTest class in the visad/examples directory.

This is only one interaction technique that can be built at an application level using VisAD. Many more are possible.

```
// create a Display
display1 = new DisplayImplJ3D("display1");

// map RealTypes to Display spatial axes
final ScalarMap map2lat = new ScalarMap(latitude, Display.YAxis);
display1.addMap(map2lat);
final ScalarMap map2lon = new ScalarMap(longitude, Display.XAxis);
display1.addMap(map2lon);
final ScalarMap map2vis = new ScalarMap(vis_radiance, Display.ZAxis);
display1.addMap(map2vis);

// link a Data object to Display
display1.addReference(ref_data, null);

// wait for Display auto-scaling (it would be more proper to wait
// until the getRange() invocations below retunr non-Missing values)
try {
  Thread.sleep(2000);
}
catch (InterruptedException e) {
}

// get ranges of values mapped to Display spatial axes
double[] range1lat = map2lat.getRange();
double[] range1lon = map2lon.getRange();
double[] range1vis = map2vis.getRange();

// create RealTuple Data objects that will be displayed at opposite
// corners of 3-D Display box
RealTuple direct_low = new RealTuple(new Real[]
                 {new Real(RealType.Latitude, range1lat[0]),
                  new Real(RealType.Longitude, range1lon[0]),
                  new Real(vis_radiance, range1vis[0])});
RealTuple direct_hi = new RealTuple(new Real[]
                 {new Real(RealType.Latitude, range1lat[1]),
                  new Real(RealType.Longitude, range1lon[1]),
                  new Real(vis_radiance, range1vis[1])});

// enable spatial axis scale displays
mode = display1.getGraphicsModeControl();
mode.setScaleEnable(true);

// color direct_low and direct_hi tuples yellow and make them
// 5 pixels wide
mode.setPointSize(5.0f);
ConstantMap[][] maps = {{new ConstantMap(1.0f, Display.Red),
                         new ConstantMap(1.0f, Display.Green),
                         new ConstantMap(0.0f, Display.Blue)}};

// link direct_low to Display with direct manipulation
final DataReferenceImpl ref_direct_low =
  new DataReferenceImpl("ref_direct_low");
ref_direct_low.setData(direct_low);
```

```
display1.addReferences(new DirectManipulationRendererJ3D(),
                       new DataReference[] {ref_direct_low}, maps);

// link direct_hi to Display with direct manipulation
final DataReferenceImpl ref_direct_hi =
  new DataReferenceImpl("ref_direct_hi");
ref_direct_hi.setData(direct_hi);
display1.addReferences(new DirectManipulationRendererJ3D(),
                       new DataReference[] {ref_direct_hi}, maps);

// construct a computational Cell that re-scales Display spatial
// axes to keep direct_low and direct_hi at corners of 3-D box
cell = new CellImpl() {
  public void doAction() throws VisADException, RemoteException {
    RealTuple low = (RealTuple) ref_direct_low.getData();
    RealTuple hi = (RealTuple) ref_direct_hi.getData();
    map2lat.setRange(((Real) low.getComponent(0)).getValue(),
                     ((Real) hi.getComponent(0)).getValue());
    map2lon.setRange(((Real) low.getComponent(1)).getValue(),
                     ((Real) hi.getComponent(1)).getValue());
    map2vis.setRange(((Real) low.getComponent(2)).getValue(),
                     ((Real) hi.getComponent(2)).getValue());
  }
};

// link cell to direct_low and direct_hi, so that its doAction
// method fires whenever the user shanges their values via
// direct manipulation
cell.addReference(ref_direct_low);
cell.addReference(ref_direct_hi);
```

Now, whenever the user tries to drag either of the yellow squares away from the corners of the 3-D box, the cell will re-scale the Display spatial axes to keep them at the corners of the box. This creates interactive scaling controls embedded in the display.

4.5 ShadowTypes

ShadowTypes are used to compute how Data objects should be displayed, given the MathType of the Data object and the ScalarMaps linked to the Display. ShadowTypes form a class hierarchy that shadows the MathType hierarchy, and a tree of ShadowTypes is constructed for each Data object to be displayed that shadows the tree of MathTypes defined for the Data object.

Furthermore, there is one ShadowType class hierarchy in the visad package, another in the visad.java3d package (all subclasses of ShadowType that adapt the corresponding class in the visad package), and presumably there will be one for each graphics API. In fact, ShadowTypes are constructed by factory methods in DataRenderer, so each DataRenderer could define a ShadowType sub-class hierarchy.

The real work of transforming Data objects into displays is done by the *doTransform* method of ShadowType. Other methods of ShadowType, such as *checkIndices* and *testIndices*, are involved in analyzing MathTypes and ScalarMaps to determine how Data objects should be transformed. It is all very complex but does define a working approach to managing the flexibility and extensibility of the VisAD visualization architecture. However,

developers do have the option of ignoring the entire structure of ShadowTypes by over-riding the *doAction* method of Display.

4.6 Displays

Display is the top-level object in the VisAD visualization architecture. Each Display object includes the following objects:

1. A window for displaying Data objects (this may be a window on a workstation screen or in virtual reality).
2. A DisplayRenderer for managing the overall rendering process.
3. A set of ScalarMaps and their associated Controls.
4. A set of DataReference objects linked to Data objects to be displayed, along with associated DataRenderers for transforming Data into display lists.

Display is actually an interface that extends the Action interface, which implements the general logic for objects that are linked to sets of DataReferences and need to be notified whenever a linked Data object changes value. Note that this may happen in two ways:

1. The Data object is mutable and its internal value changes.
2. The DataReference linked to Action is set to reference a different Data object.

Classes that implement the DisplayListener interface can be attached to Displays via their *addDisplayListener* method. Displays send DisplayEvents to attached DisplayListeners whenever certain events occur. DisplayEvents include an integer ID identifying the type of event. Event IDs include:

DisplayEvent.MOUSE_PRESSED (a mouse button is pressed),
DisplayEvent.MOUSE_PRESSED_LEFT (the left mouse button was pressed),
DisplayEvent.MOUSE_PRESSED_CENTER (the center mouse button was pressed),
DisplayEvent.MOUSE_PRESSED_RIGHT (the right mouse button was pressed),
DisplayEvent.TRANSFORM_DONE (end of transforming data objects into renderable scenes)
DisplayEvent.FRAME_DONE (end of rendering a scene)

DisplayEvents include a reference to the Display that generated them, which is either a local DisplayImpl or a RemoteDisplay for Displays on different JVMs (i.e., on different computers or on different Java interpreters on the same computer).

4.6.1 Java3D Display Constructors

The Display interface is implemented by DisplayImpl, as described in Section 6. DisplayImpl is an abstract class whose concrete subclasses are specific to particular graphics APIs. The visad.java3d package defines classes specific to the Java3D graphics API. The Java3D DisplayImpl constructors include:

```
/** construct a DisplayImpl for Java3D with a
```

```
    DefaultDisplayRendererJ3D, in a JFC JPanel */
  public DisplayImplJ3D(String name)
         throws VisADException, RemoteException;

  /** construct a DisplayImpl for Java3D with a non-default
      DisplayRenderer, in a JFC JPanel */
  public DisplayImplJ3D(String name, DisplayRendererJ3D renderer)
         throws VisADException, RemoteException;

  /** construct a DisplayImpl for Java3D with a non-default
      DisplayRenderer;
      in a JFC JPanel if api == DisplayImplJ3D.JPANEL and
      in an AppletFrame if api == DisplayImplJ3D.APPLETFRAME */
  public DisplayImplJ3D(String name, DisplayRendererJ3D renderer,
                        int api) throws VisADException, RemoteException;
```

4.6.2 Java2D Display Constructors

The Display interface is implemented by DisplayImpl, as described in Section 6. DisplayImpl is an abstract class whose concrete subclasses are specific to particular graphics APIs. The visad.java2d package defines classes specific to the Java2D graphics API. The Java2D DisplayImpl constructors include:

```
  /** construct a DisplayImpl for Java2D with a
      DefaultDisplayRendererJ2D, in a JFC JPanel */
  public DisplayImplJ2D(String name)
         throws VisADException, RemoteException;

  /** construct a DisplayImpl for Java2D with a non-default
      DisplayRenderer, in a JFC JPanel */
  public DisplayImplJ2D(String name, DisplayRendererJ2D renderer)
         throws VisADException, RemoteException;

  /** construct a DisplayImpl for Java2D with a non-default
      DisplayRenderer;
      in a JFC JPanel if api == DisplayImplJ2D.JPANEL */
  public DisplayImplJ2D(String name, DisplayRendererJ2D renderer,
                        int api) throws VisADException, RemoteException;

  /** construct a DisplayImpl for Java2D for offscreen rendering,
      with size geiven by width and height; getComponent() of this
      returns null, but display is accesible via getImage() */
  public DisplayImplJ2D(String name, int width, int height)
         throws VisADException, RemoteException;
```

4.6.3 Display Methods

Generally useful Display methods include:

```
  /** return the name of this Display; this method is inherited from
      Action */
  public String getName()
         throws VisADException, RemoteException;

  /** link map to this Display; this method may not be invoked
      after any links to DataReferences have been made */
  public void addMap(ScalarMap map)
         throws VisADException, RemoteException;
```

```
/** clear all links to maps from this Display */
public void clearMaps()
       throws VisADException, RemoteException;

/** link ref to this Display; this method may only be invoked
    after all links to ScalarMaps have been made */
public void addReference(DataReference ref)
       throws VisADException, RemoteException;

/** link ref to this Display; this method may only be invoked
    after all links to ScalarMaps have been made;
    the ConstantMap array applies only to rendering ref */
public void addReference(DataReference ref, ConstantMap[] maps)
       throws VisADException, RemoteException;

/** remove link to ref; if ref was added as part of a DataReference
    array passed to addReferences, remove links to all of them */
public void removeReference(DataReference ref)
       throws VisADException, RemoteException;

/** remove all DataReference links */
public void removeAllReferences()
       throws VisADException, RemoteException;
```

4.6.4 DisplayImpl Methods

These are methods that should only be called locally (and hence are methods of DisplayImpl rather than Display). Note also that DisplayImpl extends ActionImpl, which is described in Section 5.3. Generally useful DisplayImpl methods include:

```
/** link refs to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    the maps[i] array applies only to rendering refs[i];
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer,
                          DataReference[] refs, ConstantMap[][] maps)
       throws VisADException, RemoteException;

/** link refs to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer, DataReference[] refs)
       throws VisADException, RemoteException;

/** link ref to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    the maps array applies only to rendering ref;
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer,
                          DataReference ref, ConstantMap[] maps)
       throws VisADException, RemoteException;
```

```
/** link ref to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer, DataReference ref)
       throws VisADException, RemoteException;

/** return the JPanel or AppletPanel this DisplayImpl uses;
    returns null for an offscreen DisplayImpl */
public Component getComponent();

/** return a captured image of the display */
public BufferedImage getImage();

/** return the DisplayRenderer associated with this DisplayImpl */
public DisplayRenderer getDisplayRenderer();

/** return the ProjectionControl associated with this DisplayImpl */
public ProjectionControl getProjectionControl();

/** return the GraphicsModeControl associated with this DisplayImpl */
public GraphicsModeControl getGraphicsModeControl();

/** add a DisplayListener */
public void addDisplayListener(DisplayListener listener);

/** remove a DisplayListener */
public void removeDisplayListener(DisplayListener listener);

/** re-apply auto-scaling of ScalarMap ranges next time
    Display is triggered */
public void reAutoScale();

/** if auto is true, re-apply auto-scaling of ScalarMap ranges
    every time Display is triggered */
public void setAlwaysAutoScale(boolean auto);

/** disable the action of this DisplayImpl, but respond to any
    accumulated events after it is re-enabled; this method does
    not return until actions have ceased in this DisplayImpl */
public void disableAction();

/** re-enable this previously disabled DisplayImpl, and respond
    to any accumulated events */
public void enableAction();

/** create a projection matrix appropriate for this graphics API
    from x, Y and Z rotation angles, from a scale factor, and from
    X, Y and Z translation amounts;
    these creates the matrix format used by the getMatrix and
    setMatrix methods of ProjectionControl;
    note DisplayImplJ3D returns an array of length 16 (4 x 4 matrix)
    and DisplayImplJ2D returns an array of length 6 (2 x 3 matrix) */
public double[] make_matrix(double rotx, double roty, double rotz,
                            double scale, double transx,
                            double transy, double transz);

/** return the product of matrices a and b, according to the matrix
    format for this graphics API */
```

```
public double[] matrix_multiply(double[] a, double[] b);

/** wait for milliseconds; this is deprecated, use
    'new visad.util.Delay(milliseconds)' instead */
public static void delay(int milliseconds)
       throws VisADException;
```

4.6.5 RemoteDisplayImpl Methods

These are methods that should only be called locally (and hence are methods of RemoteDisplayImpl rather than RemoteDisplay). Generally useful RemoteDisplayImpl methods include:

```
/** link refs to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    the maps[i] array applies only to rendering refs[i];
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer,
                          DataReference[] refs, ConstantMap[][] maps)
       throws VisADException, RemoteException;

/** link refs to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer, DataReference[] refs)
       throws VisADException, RemoteException;

/** link ref to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    the maps array applies only to rendering ref;
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer,
                          DataReference ref, ConstantMap[] maps)
       throws VisADException, RemoteException;

/** link ref to this Display using the non-default renderer;
    this method may only be invoked after all links to ScalarMaps
    have been made;
    this is a method of DisplayImpl and RemoteDisplayImpl rather
    than Display - see Section 6.1 for more information */
public void addReferences(DataRenderer renderer, DataReference ref)
       throws VisADException, RemoteException;
```

4.6.6 DisplayListener Methods

DisplayListener is an interface that extends EventListener.

```
/** send a DisplayEvent to this DisplayListener */
public void displayChanged(DisplayEvent event)
       throws VisADException, RemoteException;
```

4.6.7 DisplayEvent Methods

DisplayEvent is a class that extends Event.

```
/** get the DisplayImpl that sent this DisplayEvent (or
    a RemoteDisplay reference to it if the Display was on
    a different JVM) */
public Display getDisplay();

/** get the ID type of this event; legal ID's are
    DisplayEvent.MOUSE_PRESSED, DisplayEvent.MOUSE_PRESSED_CENTER
    DisplayEvent.TRANSFORM_DONE and DisplayEvent.RENDER_DONE */
public int getId();

/** get the window X coordinate if this is a mouse pressed event */
public int getX();

/** get the window Y coordinate if this is a mouse pressed event */
public int getY();
```

4.7 Shapes

The DisplayRealType Display.Shape can be a very powerful tool for building complex displays. When RealTypes are mapped to Display.Shape, their values are quantized according to the Set argument to ShapeControl.setShapeSet and the resulting indices are used to look up VisADGeometryArray shapes. These are located according to any RealTypes mapped to spatial DisplayRealTypes, scaled according to any RealType mapped to Display.ShapeScale, and for those VisADGeometryArrays that do not include color values, colored according to any RealTypes mapped to color DisplayRealTypes. Multiple RealTypes may be mapped to Display.Shape, allowing the creation of composite shapes with different sub-shapes determined by values of different RealTypes. Also, the same RealType may be mapped to Display.Shape more than once (Display.Shape is the only DisplayRealType for which this is possible) in order to allow composite shapes that combine lines (e.g., VisADLineArrays) and surfaces (e.g., VisADTriangleArrays).

The PlotText.render_label method, documented in Section 4.7.2, is useful for generating shapes from text Strings. The ShapeControl.setShapeSet method can be used to define a quantization of real values, and shapes can be generated from numerical strings of quantization values. The start argument to PlotText.render_label method can be used to generate different offsets for text plots of values of different RealTypes. This could be used to generate traditional station plots from meteorological data.

If the Set argument to setShapeSet has length = 1 (i.e., just one member) then all RealType values map to the VisADGeometryArray shape at index = 0.

For examples on how to use Shape, see examples/DisplayTest.java cases 46 and 47.

4.7.1 VisADGeometryArray Shapes

VisADGeometryArray is an abstract class that defines public variables for describing a 3-D shape (or a 2-D shape if Z valaues are ignored). The variables are:

```
public int vertexCount;
public float[] coordinates;
public float[] normals;
public byte[] colors;
```

Only vertexCount and coordinates must be set, and the length of coordinates must be 3 times vertexCount. Each group of 3 coordinates values defines the XAxis, YAxis and ZAxis (ignored for Java2D displays) spatial coordinates of a vertex. Spatial coordinates vary from -1.0f to +1.0f in the VisAD display "box". In general, the coordinates of shapes should be centered around the origin (0.0f, 0.0f, 0.0f) and scaled appropriately relative to the "box" dimensions. Use RealTypes mapped to spatial DisplayRealTypes to determine absolute shape locations, and RealTypes mapped to ShapeScale to determine relative shape sizes.

For triangles or quads in Java3D, normals must be set and have the same length as coordinates. Normals should be normalized to length 1.0f.

If colors is set its length must be 3 times vertexCount for Red, Green and Blue color components or 4 times vertexCount to also include Alpha (transparency, in Java3D only). Colors values are unsigned bytes because these are used by Java3D, but NOTE that unsigned bytes are not a supported primitive type of Java. If colors is not set, shape colors are determined by RealTypes mapped to color DisplayRealTypes.

The subclasses of VisADGeometryArray are:

VisADPointArray - each vertex is rendered as a point
VisADLineArray - each pair of vertices is rendered as a line
VisADTriangleArray - each three vertices is rendered as a triangle
VisADQuadArray - each four vertices is rendered as a quadrangle
VisADLineStripArray - see description below
VisADTriangleStripArray - see description below
VisADIndexedTriangleStripArray - see description below

These classes all have no-argument constructors, relying on public direct access to their variables. These classes behave like the corresponding classes (just remove "VisAD" from the class name) in the javax.media.j3d package (i.e., Java3D). Note that for VisADLineArray, VisADTriangleArray and VisADQuadArray vertexCount must be a multiple of 2, 3 and 4 respectively. For VisADLineStripArray, VisADTriangleStripArray and VisADIndexedTriangleStripArray sequences of vertices are rendered as strips of lines and triangles, as described for the LineStripArray, TriangleStripArray and IndexedTriangleStripArray classes in Java3D.

4.7.2 The PlotText.render_label Method

The PlotText.render_label method is useful for generating VisADLineArray shapes from text Strings, for use with ShapeControl. It is a static method as follows:

```
/** create a VisADLineArray rendering of string, located at
    start, with characters separated by base and character
    vertical in the up direction (start, base and up are all
    double[3] arrays); center text at start if center is true */
public static VisADLineArray render_label(String string,
       double[] start, double[] base, double[] up, boolean center);
```

5. Computational Cells

Cell, like Display, is an interface that extends Action. A Cell object defines a computation that is triggered whenever any of its linked Data object changes. Cells can be used to implement spread sheet cells that are recomputed when values of other cells change (this is the source of our use of the name Cell). Cells can also be used to implement data flow networks. (Another possible extension of Action could be defined for a link in a store and forward data distribution network, such as implemented by the Unidata Program for distributing meteorological data to universities [1].)

The VisAD system does not include class hierarchies for defining computations, the way it does for defining Data and Displays. This is because the Java programming language defines an adequate set of structures for defining computations, including the ability to link to functions written in other languages (e.g., C and Fortran) via the Java Native Interface (JNI).

However, the VisAD Data classes do define methods for basic arithmetical and mathematical operations. These include all the Java primitive operations (e.g., add, subtract) and the operations of the java.lang.Math class (e.g., sqrt, sin, max), as described in Section 3.2.2. They also include operations specific to Data subclasses, such as Tuple component access and Function evaluation and resampling, as described in Sections 3.2.5 and 3.2.7.

5.1 Cell Constructors

Cell is an interface that may apply to both local and remote Cell objects. CellImpl is an abstract class that only applies to local Cell objects, and RemoteCell is an interface that only applies to remote Cell objects (see Section 6 for more information). Developers extend CellImpl to define new computations and may invoke the following super constructors:

```
public CellImpl();

/** the name String can be useful for debugging */
public CellImpl(String name);
```

5.2 Cell Methods

```
/** return the name of this Cell; this method is inherited from
    Action */
public String getName()
       throws VisADException, RemoteException;

/** this defines the computation performed by this Cell;
    it is invoked whenever linked Data objects change */
public abstract void doAction()
       throws VisADException, RemoteException;

/** link ref to this Cell */
public void addReference(DataReference ref)
       throws VisADException, RemoteException;

/** remove link to ref */
public void removeReference(DataReference ref)
       throws VisADException, RemoteException;
```

```
  /** remove all DataReference links */
  public void removeAllReferences()
         throws VisADException, RemoteException;

  /** set a non-triggering link to a DataReference; this is
      used to give the Cell access to Data without triggering
      the Cell's doAction whenever the Data changes;
      these 'other' DataReferences are identified by their
      integer index */
  public void setOtherReference(int index, DataReference ref)
         throws VisADException, RemoteException;

  /** return the non-triggering link to a DataReference
      identified by index */
  public DataReference getOtherReference(int index)
         throws VisADException, RemoteException;
```

5.3 ActionImpl Methods

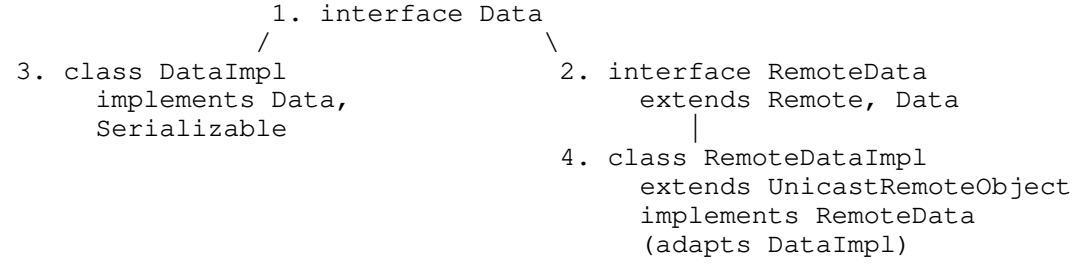
CellImpl and DisplayImpl both extend ActionImpl, and share the following methods:

```
  /** disable the action of this CellImpl, but respond to any
      accumulated events after it is re-enabled; this method does
      not return until actions have ceased in this CellImpl */
  public void disableAction();

  /** re-enable this previously disabled CellImpl, and respond
      to any accumulated events */
  public void enableAction();

  /** increase the current maximum limit on the number of Java
      Threads used for all ActionImpls; the default maximum is 10
      and num cannot be less than the current maximum */
  public void setThreadPoolMaximum(int num)
         throws Exception;
```

6. Distributed Computing

VisAD uses the Java Remote Method Invocation (RMI) API for distributed computing. RMI allows Java objects on remote machines to be accessed with the same syntax used to access local objects. VisAD exploits this so that its low-level logic can be applied to remote objects transparently. On the other hand, application developers can control the distinction between local and remote objects in order to properly manage performance and Exception handling.

In order to adapt to RMI, the Data class hierarchy is replicated four times:



1. As interfaces (Data, Function, Field, etc.) that are implemented by both local and remote Data classes.
2. As interfaces (RemoteData, RemoteFunction, RemoteField, etc.) that extend those in 1 and extend java.rmi.Remote, and are only implemented by remote Data classes. Not all Data sub-classes have Remote interfaces.
3. As local Data classes (DataImpl, FunctionImpl, FieldImpl, etc.) that implement the interfaces in 1 (but not those in 2) and implement Serializable.
4. As remote Data classes (RemoteDataImpl, RemoteFunctionImpl, RemoteFieldImpl, etc.) that extend java.rmi.server.UnicastRemoteObject and implement the interfaces in 2. These remote implementations are simple adapters for the corresponding local implementations (i.e., the classes in 3), except that some methods check that local implementations get local arguments and remote implementations get remote arguments. Not all Data sub-classes have remote implementations.

The low-level logic of VisAD uses the interfaces in 1 that apply to both local and remote Data objects. Specifically, method arguments and return values are declared with the interfaces in 1. When methods are invoked on remote objects Java can decide at run time whether to pass arguments and return values by copy or by remote reference, depending on the whether actual argument and return value objects implement Serializable (the classes in 3) or Remote (the interfaces in 2). This is fundamentally important because:

* Application developers have the freedom to use remote objects wherever they like.

This same replication of classes into four distinct hierarchies is also applied to DataReference (i.e., DataReference, RemoteDataReference, DataReferenceImpl, RemoteDataReferenceImpl) and to the Action class hierarchy (which includes Display). This allows Displays to be linked to remote DataReference objects to support remote visualization, and allows connections between remote Displays to support collaborative visualization. It is even possible that the components of a Tuple or the range samples of a Field may reside on multiple remote machines - note however that application developers should exploit such freedom carefully.

When developers need to distinguish between local and remote objects, local objects can be accessed using the classes in 3, and remote objects can be accessed using the interfaces in 2. Objects that are going to accessed remotely should use the constructors of the classes in 4, but declared using the interfaces in 1 or 2.

6.1 Distributed Computing Guidelines and Cautions

The easiest way to develop distributed and collaborative applications is to copy the patterns in the GoesCollaboration application described in Section 12.3 with complete listing in Appendix B. This section discusses the general guidelines for designing distributed and collaborative applications, and a few cautions about ways that programming in a distributed environment differs from the non-distributed environment.

The *addReference* method of Display and Cell (inherited from Action) is invoked by applications to create the network of Data, Display and computational Cell objects. When

the *addReference* method is invoked for RemoteDisplays and RemoteCells, the arguments should be instances of RemoteDataReference. This is because a local DataReferenceImpl will be passed by copy and the RemoteDisplay or RemoteCell will be linked with the copy rather than the intended DataReference. Applications can easily construct RemoteDataReferenceImpl objects for any local DataReferenceImpl objects that they need to link to RemoteDisplays or RemoteCells. Similarly, when the *addReference* method is invoked for local DisplayImpl and CellImpl objects, the argument should be a local DataReferenceImpl object. The general rule is:

* Connect local to local and remote to remote using *addReference*.

In contrast, the *addReferences* method of DisplayImpl and RemoteDisplayImpl can accept a mix RemoteDataReferenceImpl and local DataReferenceImpl arguments. However, the *addReferences* method is not defined for the RemoteDisplay interface (or for the Display interface) and hence may not be invoked remotely. It can only be invoked locally so local DataReferenceImpl arguments are not copied. That is, the *addReferences* method can be used to create links between Data and Displays that are on different JVMs (i.e., different computers or different Java interpreters running on the same computer), but may only be invoked on the JVM of the Display.

The rule about connecting local to local and remote to remote also applies to the *setData* method of DataReference that creates links between DataReference objects and Data objects. In particular, only local DataImpl objects may be the argument of the *setData* method of a local DataReferenceImpl object. However, both local and remote Data objects may be argument of the *setData* method of a RemoteDataReference object. This is because many Data subclasses can only be local (e.g., Real, RealTuple, Tuple, Set). Note, however, that when a local DataImpl object is the argument of the *setData* method of a RemoteDataReference object, then a copy of that argument is passed to the JVM of the RemoteDataReference object. This can lead to the following problem if the local DataImpl argument is mutable (i.e., a FieldImpl or a FlatField): the application may modify the local DataImpl object but these changes will not be reflected in the copy that is actually linked to the RemoteDataReference object. Developers of distributed applications should remember that:

* Local FieldImpl and FlatField arguments to RemoteDataReference.*setData* are dangerous.

A Data object may have sub-objects residing on multiple JVMs. This is because the component arguments to the Tuple constructor and the range sample arguments to the FieldImpl constructor are declared as Data and may be either local or remote. This should be used with care. It can result in very poor performance. Furthermore, data modification events are not propagated from sub-objects to parent objects on different JVMs.

The Data, DataReference, Display and Cell classes all support remote access. However, only the Data class and its associated metadata classes also support copying between JVMs. Thus DataReference, Display and Cell objects are fixed to the machine where they are created (although their methods can be invoked remotely). Copying Display

objects makes no sense, since they are attached to a physical display device. Cell objects should not be copied between JVMs since they may include calls to functions written in other programming languages which are not generally portable between machines. Copying DataReferences is dangerous because they define data identities in applications.

Developers should think about distributed applications as consisting of DataReference, Display, Cell and user interface objects with fixed locations, and which communicate by exchanging Data and ThingChangedEvents (ThingChangedEvents are invisible to applications - they are used to notify Displays and Cells when Data values chnage).

Finally we offer a few general cautions for programming with distributed objects. First, beware of static variables that are not constant across all JVMs in any Serializable class. When objects are copied to new JVMs they will encounter new values for non-constant static variables. For example, if you want to enforce that every instance of a class have a unique name String, you would do this with a static Vector of names. But when instances are copied between JVMs there is no way to enforce that names are unique.

Similarly, beware of using '==' or '!=' tests between instances of any Serializable class. Rather, use the *equals* method and explicitly define the conditions for equality. For example, an object should probably be equal to a clone of itself.

6.2 Connecting to Remote Machines

In order for applications to communicate with applications on other JVMs they need a way to obtain remote references to objects on those JVMs. Java RMI provides ways to:

1. Bind an object to a URL using java.rmi.Naming.rebind(String url, Object obj).
2. Obtain a remote reference to an object bound to a URL using java.rmi.Naming.lookup(String url).

Rather than requiring applications to bind each remote object to a URL, the VisAD system provides the RemoteServer interface and the RemoteServerImpl class for serving and accessing arrays of RemoteDataReference objects. An application can bind one RemoteServerImpl objct to a URL and then use it to serve many RemoteDataReference objects to applications on other JVMs.

The GoesCollaboration application described in Section 12.3 and listed in Appendix B includes examples of how RemoteServerImpl and java.rmi.Naming should be used.

Note that remote implementation classes (in VisAD these have names matching Remote*Impl) require a second compilation step to generate RMI stub and skel classes. In JDK this second compilation step is done with the rmic compiler.

Note also that an RMI server must be running on a machine where applications invoke the java.rmi.Naming.rebind method. In JDK this RMI server is rmiregistry.

6.2.1 RemoteServerImpl Constructors

Construct a RemoteServerImpl to serve RemoteDataReference to remote JVMs.

```
/** construct a RemoteServerImpl and initialize it with
    an array of RemoteDataReferenceImpls */
public RemoteServerImpl(RemoteDataReferenceImpl[] refs)
```

```
        throws RemoteException;
```

6.2.2 RemoteServer Methods

These methods are used to remotely (or locally) access RemoteDataReference objects from a RemoteServer.

```
/** return the RemoteDataReference with index on this
    RemoteServer, or null */
public RemoteDataReference getDataReference(int index)
       throws RemoteException;

/** return the RemoteDataReference with name on this
    RemoteServer, or null */
public RemoteDataReference getDataReference(String name)
       throws VisADException, RemoteException;

/** return an array of all RemoteDataReferences on this
    RemoteServer, or null */
public RemoteDataReference[] getDataReferences()
       throws VisADException, RemoteException;
```

6.2.3 RemoteServerImpl Methods

These methods are used to set RemoteDataReference objects to be served.

```
/** set one RemoteDataReference in the array on this
    RemoteServer (and extend length of array if necessary) */
public void setDataReference(int index, RemoteDataReference ref)
       throws VisADException;

/** set array of all RemoteDataReferences on this RemoteServer */
public void setDataReferences(RemoteDataReference[] refs);
```

6.3 Application Example: Collaborative Direct Manipulation

In this example users at different workstations visualize and re-draw the same three Data objects: a Real object, a RealTuple object and a FlatField object. The server code constructs the Data objects and their DataReferences which it links to a Display via a DirectManipulationRenderJ3D. It also constructs a RemoteServer for the DataReferences, and binds it to a URL. The client code looks up the RemoteServer via the URL, uses it to get the DataReferences, and links them to a Display via a DirectManipulationRenderJ3D. These examples are based on the DisplayTest class. Here is the server code:

```
// construct three Data objects,
FunctionType field_type = new FunctionType(reala, realb);
FlatField field = FlatField.makeField(field_type, 64, false);
Real real = new Real(reala, 2.0);
Real[] reals3 = {new Real(reala, 1.0), new Real(realb, 2.0),
                 new Real(realc, 1.0)};
RealTuple real_tuple = new RealTuple(reals3);

// construct a Display
display1 = new DisplayImplJ3D("display1");

// map RealTypes to Display spatial axes
display1.addMap(new ScalarMap(reala, Display.XAxis));
```

```
display1.addMap(new ScalarMap(realb, Display.YAxis));
display1.addMap(new ScalarMap(realc, Display.ZAxis));

// 5 pixel size for Real and RealTuple objects
mode = display1.getGraphicsModeControl();
mode.setPointSize(5.0f);

// construct DataReferences for three Data objects and link them
// to Display via DirectManipulationRendererJ3Ds
ref_real = new DataReferenceImpl("ref_real");
ref_real.setData(real);
display1.addReferences(new DirectManipulationRendererJ3D(),
                       new DataReference[] {ref_real});

ref_real_tuple = new DataReferenceImpl("ref_real_tuple");
ref_real_tuple.setData(real_tuple);
display1.addReferences(new DirectManipulationRendererJ3D(),
                       new DataReference[] {ref_real_tuple});

ref_field = new DataReferenceImpl("ref_field");
ref_field.setData(field);
display1.addReferences(new DirectManipulationRendererJ3D(),
                       new DataReference[] {ref_field});

// create RemoteDataReferences
RemoteDataReferenceImpl[] rem_data_refs =
  new RemoteDataReferenceImpl[3];
rem_data_refs[0] = new RemoteDataReferenceImpl(ref_field);
rem_data_refs[1] = new RemoteDataReferenceImpl(ref_real);
rem_data_refs[2] = new RemoteDataReferenceImpl(ref_real_tuple);

// construct a RemoteServer for the RemoteDataReferences
RemoteServerImpl obj = new RemoteServerImpl(rem_data_refs);

// and bind it to a URL
Naming.rebind("//:/RemoteServerTest", obj);
```

Once the server is running, any number of clients can connect and share access to the same set of three Data objects. Here is the client code:

```
// lookup RemoteServer by URL specified in domain String
RemoteServer remote_obj = (RemoteServer) Naming.lookup(domain);

// get three RemoteDataReferences from RemoteServer
RemoteDataReference field_ref = remote_obj.getDataReference(0);
RemoteDataReference real_ref = remote_obj.getDataReference(1);
RemoteDataReference real_tuple_ref = remote_obj.getDataReference(2);

// get RealTupleType of real_tuple Data object
dtype = (RealTupleType) real_tuple_ref.getData().getType();

// construct a Display
display1 = new DisplayImplJ3D("display");

// map RealType components of real_tuple to Display spatial axes
display1.addMap(new ScalarMap((RealType) dtype.getComponent(0),
                              Display.XAxis));
```

```
display1.addMap(new ScalarMap((RealType) dtype.getComponent(1),
                              Display.YAxis));
display1.addMap(new ScalarMap((RealType) dtype.getComponent(2),
                              Display.ZAxis));

// 5 pixel size for Real and RealTuple objects
mode = display1.getGraphicsModeControl();
mode.setPointSize(5.0f);

// construct RemoteDisplay to link to RemoteDataReferences
// (recall that we must connect remote to remote)
RemoteDisplayImpl remote_display1 = new RemoteDisplayImpl(display1);

remote_display1.addReferences(new DirectManipulationRendererJ3D(),
                              new DataReference[] {real_ref});

remote_display1.addReferences(new DirectManipulationRendererJ3D(),
                              new DataReference[] {real_tuple_ref});

remote_display1.addReferences(new DirectManipulationRendererJ3D(),
                              new DataReference[] {field_ref});
```

7. File Format and Data Form Adapters

Data form adapters take an identifier for a external data object (i.e., external to VisAD), such as a fully qualified file name or a URL, and return a VisAD Data object. The most common data forms are file formats, but data forms may include any other source of data. Data form adapters provide access to data and metadata via the VisAD Data and metadata APIs. Adapters include transparent management of data movement between memory and their native storage medium (e.g., disks for files), although developers may extend the CachingStrategy class to define their own data migration policy. The initial release of VisAD only provides transparent data management for HDF-EOS files.

A given file or other data object can generally be interpreted as many different VisAD MathTypes. For example, data with the following MathType:

```
(time -> (temperature, pressure))
```

could also be read as:

```
((time -> temperature), (time -> pressure))
```

The first MathType is usually preferable, since it makes it clear that the temperature and pressure Fields have the same time sampling. However, in some cases the second MathType may be preferred, for example to combine this with other data having the second MathType (and possibly temperatures and pressures with unequal time samplings).

Similarly, data with the MathType:

```
(latitude -> (longitude -> pressure))
```

could also be read as:

```
((latitude, longitude) -> pressure)
```

Again, the first MathType is usually preferable, since it makes it clear that the domain sampling Set of (latitude, longitude) can be factored into a product of latitude samples and longitude samples. However, the second MathType may be preferable in order to combine this data with other data whose (latitude, longitude) sampling cannot be factored, or whose domain MathType is (row, column) with a CoordinateSystem whose Reference is (latitude, longitude).

Thus our approach is to develop a variety of adapters for each data format, in order to give application developers and end users a choice of how to interpret data in terms of the VisAD data model. However, in the initial release of VisAD, however, there is only a single adapter per data format.

Adapters initially exist for FITS, netCDF, HDF-EOS, GIF and Vis5D file formats. The contacts for help with each file format are:

```
FITS      Dave Glowacki   dglo@ssec.wisc.edu
netCDF    Steve Emmerson  steve@unidata.ucar.edu
HDF-EOS   Tom Rink        rink@ssec.wisc.edu
GIF       Dave Glowacki   dglo@ssec.wisc.edu
Vis5D     Bill HIbbard    hibbard@facstaff.wisc.edu
```

7.1 Extracting Metadata From Data Objects Returned by Data Form Adapters

When applications explicitly construct Data objects they must also explicitly construct their MathTypes and other metadata and so "know" the value of those metadata. In contrast, Data objects returned by data form adapters are constructed internally by those adapters, often using metadata from stored data objects, and application must extract the MathTypes and other metadata of Data objects returned by adapters.

The MathType of any Data object is returned by the *getType* method of Data. MathTypes have tree structures that can be recursively "parsed" with code like:

```
MathType type = data.getType();
if (type instanceof FunctionType) {
  RealTupleType domain = ((FunctionType) type).getDomain;
  MathType range = ((FunctionType) type).getRange();
  // recursively analyze domain and range
}
else if (type instanceof RealTupleType) {
  int dimension = ((TupleType) type).getDimension();
  RealType[] types = new RealType[dimension];
  for (int i=0; i<dimension; i++) {
    types[i] = (RealType) ((TupleType) type).getComponent(i);
  }
  // recursively analyze types
}
else if (type instanceof TupleType) {
  int dimension = ((TupleType) type).getDimension();
  MathType[] types = new MathType[dimension];
  for (int i=0; i<dimension; i++) {
    types[i] = ((TupleType) type).getComponent(i);
  }
  // recursivley analyze types
}
else if (type instanceof RealType) {
```

```
    // this is a leaf in the MathType "tree"
    // map it to a DisplayRealType, get its default Unit, etc
  }
  else if (type insatnceof TextType) {
    // this is a leaf in the MathType "tree"
  }
```

Most applications will try to fit MathTypes into broad categories, such as "image", "grid" or "table". Data displays are defined by ScalarMaps (described in Section 4.1) involving the RealTypes that are extracted from the MathTypes of Data to be displayed. Applications may define general policies for constructing ScalarMaps for each broad category of MathTypes. Section 4.1.1 describes some general guidelines for defining ScalarMaps.

Other metadata such as Units, CoordinateSystems, Sets, ErrorEstimates and missing data indicators can be extracted from Data objects using methods such as:

```
boolean Data.isMissing()

Unit Real.getUnit()
ErrorEstimate Real.getError()

Unit[] RealTuple.getTupleUnits()
CoordinateSystem RealTuple.getCoordinateSystem()
ErrorEstimate[] RealTuple.getErrors()

Set Field.getDomainSet()
Unit[] Field.getDomainUnits()
CoordinateSystem Field.getDomainCoordinateSystem()

Unit[] FlatField.getRangeUnits()
CoordinateSystem FlatField.getRangeCoordinateSystem()
CoordinateSystem FlatField.getRangeCoordinateSystem(int index)
ErrorEstimate[] FlatField.getRangeErrors()
```

These methods are documented in the appropriate sub-sections of Section 3.2.

7.2 General Design of Data Form Adapters

Data form and file format adapters extend the abstract class Form in the visad.data package. Just as adapters allows data stored in different formats to be accessed via the uniform API of the VisAD Data and metadata classes, the Form class provides a uniform API for higher-level data access operations such as open.

Each file format may include multiple sub-classes of Form, each defining a different policy for how data objects are adapted to the VisAD Data and metadata classes. For example, different Form sub-classes may return Data objects with different MathTypes for the same external data object.

The general design for data form adapters is unfinished, and will be further elaborated in later versions of VisAD. Furthermore, the functionality of current adapters varies between different file formats.

7.2.1 Form Methods

Useful Form methods include:

```
/** open a data object specified by a String id, commonly a
    file name, and return a DataImpl that adapts it to the
    VisAD Data and metadata API */
public DataImpl open(String id)
       throws BadFormException, IOException, VisADException;

/** open a data object specified by a URL, and return a DataImpl
    that adapts it to the VisAD Data and metadata API */
public DataImpl open(URL url)
       throws BadFormException, IOException, VisADException;

/** store data in an external data object specified by a
    String id, commonly a file name; only over-write an
    existing data object if replace is true */
public void save(String id, Data data, boolean replace)
       throws BadFormException, IOException, VisADException,
              RemoteException;
```

7.3 FITS Adapter

The FITS file adapter is defined in the visad.data.fits package. It includes one sub-class of Form, FitsForm, which only implements the *open(String id)* method. This can generally adapt primary images, image extensions and binary tables. It does not initially adapt ASCII tables. The returned Data object simply omits any parts of FITS files that FitsForm cannot adapt. We want to thank Tom McGlynn of NASA for his help.

FitsForm has the constructor:

```
public FitsForm();
```

A FitForm instance can open any number of FITS files.

7.4 netCDF Adapter

The netCDF file adapter is defined in the visad.data.netcdf and visad.data.netcdf.units packages. It includes one sub-class of Form, Plain, which implements the *open(String id)* and *save(String id, Data data, boolean replace)* methods. We want to thank Russ Rew and Glenn Davis of the Unidata Program Office for their help.

Plain has the constructor:

```
public Plain();
```

A Plain instance can open and save any number of netCDF files.

Plain leaves most netCDF arrays unfactored. However, it will factor netCDF arrays whose outermost dimension is time (recognized by units convertable with seconds, or by the name 'time'). Thus, rather than returning a Data object with the MathType:

```
((time, latitude, longitude, altitude) -> temperature)
```

it will factor this into:

```
(time -> ((latitude, longitude, altitude) -> temperature))
```

This permits time to be mapped to Display.Animation (it could not be mapped to Animation in the unfactored MathType because the Display could not be guaranteed that it will be able to factor a Set of time samples for animation steps from the unfactored MathType).

7.5 HDF-EOS Adapter

The HDF-EOS file adapter is defined in the visad.data.hdfeos and visad.data.hdfeos.hdfeosc packages. It includes one sub-class of Form, HdfeosDefault, which only implements the *open(String id)* method. This can generally adapt grid and swath data, but not point data. Since the HDF-EOS file format definition is still changing and little data is available in HDF-EOS format, our HDF-EOS file adapter is still unstable, particularly in its handling of swath metadata. Polar stereo and Lambert conformal CoordinateSystems are defined for grid data, but grid data in other coordinate systems are not initially geo-referenced (i.e., they are returned with row-column domains that do not define any CoordinateSystem relative to latitude-longitude). We want to thank Mike Jones of NASA for his help.

The HDF-EOS file adapter invokes native methods so its installation includes special procedures for creating a shared object file.

HdfeosDefault has the constructor:

```
public HdfeosDefault();
```

An HdfeosDefault instance can open any number of HDF-EOS files.

The VisAD HDF-EOS fila adapter is dependent on software that must be obtained from NASA and NCSA. Specifically, users must obtain and install HDF4.1r1 from:

```
ftp://ftp.ncsa.uiuc.edu/HDF/HDF_Current
```

then obtain and install HDF-EOS:

```
http://ulabibm.gsfc.nasa.gov/hdfeos/hdf.html#4
```

7.6 GIF / JPEG Adapter

The GIF / JPEG file adapter is defined in the visad.data.gif package. It includes one sub-class of Form, GIFForm, which implements the *open(String id)* and *open(URL url)* methods. These *open* methods always return a FlatField with MathType:

```
((ImageElement, ImageLine) -> (Red, Green, Blue))
```

GIFForm has the constructor:

```
public GIFForm();
```

A GIFForm instance can open any number of GIF and JPEG files.

7.7 Vis5D Adapter

The Vis5D file adapter is defined in the visad.data.vis5d package. It includes one sub-class of Form, Vis5DForm, which implements the *open(String id)* method. The initial implementation will only open files where all fields have the same number of vertical levels, and the returned Data objects do not include CoordinateSystems for geo-referencing data.

The Vis5D file adapter invokes native methods so its installation includes special procedures for creating a shared object file.

Vis5DForm has the constructor:

```
public Vis5DForm();
```

A Vis5DForm instance can open any number of Vis5D files.

7.8 McIDAS Adapter

The McIDAS file adapter is defined in the visad.data.mcidas package. It includes one sub-class of Form, AreaForm, which implements the *open(String id)* and *open(URL url)* methods. AreaForm is designed for McIDAS area files (i.e., image files). For GVAR area files, the returned Data objects include CoordinateSystems that implement GVAR navigation. The *open(URL url)* method uses custom URLs for McIDAS ADDE servers.

AreaForm has the constructor:

```
public AreaForm();
```

An AreaForm instance can open any number of McIDAS area files.

7.9 VisAD Adapter (serialized Java objects)

The VisAD file adapter is defined in the visad.data.visad package. It includes one sub-class of Form, VisADForm, which implements the *open(String id)*, *open(URL url)*, and *save(String id, Data data, boolean replace)* methods. VisAD files are simply DataImpl objects turned into byte streams by Java serialization.

VisADForm has the constructor:

```
public VisADForm();
```

A VisADForm instance can open any number of VisAD files.

Note that the *main* method of VisADForm can be used to convert any VisAD-readable file to a VisAD file. The command:

```
java -mx64m visad.data.visad.VisADForm in_file out_file.vad
```

will read in_file in any VisAD-readable format and convert it to a VisAD file in out_file.vad (note that ".vad" is the standard file name extension for VisAD files).

Note that VisAD classes do not yet implement version IDs and that VisAD class implementations are still changing. Thus serialized VisAD DataImpl objects are not appropriate for long term data storage.

8. User Interfaces

The primary lesson learned from the C implementation of VisAD, and from experience with other general visualization systems, is that user interfaces should not reflect the full generality of the underlying system. The power of VisAD comes from providing a context in which developers can answer questions like "what is the nature of an image?" However, end users should not be required to answer such questions in order to manipulate and visualize their images.

Thus VisAD is designed to support a wide variety of user interfaces that present choices in terms that are familiar to users. Our specific plans for user interface experiments include:

1. A customizable data viewer applet that data providers can embed in their web pages to provide browsers with interactive 3-D visualizations of their data. Most decisions would be made by data providers so that end-user choices are simple (e.g., select data, animate, rotate view). This applet would allow multiple browsers to share their choices for collaborative data visualization.
2. A spread-sheet with a Data object and Display of that Data object in each Cell [4]. Some of these Data objects would be read from files, some would be defined by the user via direct manipulation, and some would be computed from other Data objects by simple formulas or by Java programs. Such spread-sheets would be very useful, particularly with a facility for saving and editing the spread-sheet configuration (i.e., the MathTypes of each Cell's Data, the ScalarMaps of each Cell's Display, and the files, direct manipulation DataRenderers, formulas and Java programs that define each Cell's Data values). Multiple users may share the same spread-sheet configuration for collaborative development of data analysis algorithms. The application described in Section 12.3 is a simple collaborative spreadsheet.
3. A web browser JavaBean and a drag-and-drop interface for copying data found on the web into VisAD Data objects for input to users' data analysis programs. Of course, this will only work for data in formats that are adapted to VisAD Data classes (see Section 7).
4. Experiments with interactive visualization techniques appropriate for highly spectral satellite data (i.e., hundreds or thousands of spectral channels).
5. Implementations of existing visualization user interfaces on top of VisAD, such as Vis5D.

8.1 VisAD User Interface Classes

Extensive libraries of user interface classes are available in the java.awt.swing packages (also known as the Java Foundation Classes or JFC) and these work well with VisAD. NCSA's Habanero is very useful for building distributed and collaborative user interfaces for use with VisAD. Information about Habanero is available at:

```
http://www.ncsa.uiuc.edu/SDG/Software/Habanero/
```

The visad.util package includes classes for needed user interface components that are not included in JFC, and for extensions of JFC classes that include connections to VisAD objects.

8.1.1 VisADSlider Constructor

The VisADSlider class extends java.awt.swing.JPanel. It includes a JSlider, a JLabel, a DataReference to a Real, and a Cell. If either the JSlider or Real changes value the VisADSlider updates the other. The JLabel shows the current value.

Several VisADSliders on different JVMs may be connected to the same RemoteDataReference to create a collaborative user interface slider.

```
/** JSlider values range between low and hi (with initial value
    start) and are multiplied by scale to create Real values
    of type referenced by ref */
public VisADSlider(String name, int low, int hi, int start,
                   double scale, DataReference ref, RealType type)
       throws VisADException, RemoteException;
```

8.1.2 LabeledRGBWidget and LabeledRGBAWidget Constructors

The LabeledRGBWidget and LabeledRGBAWidget classes extend java.awt.Panel. They provides a way for users to interactively change pseudo-color lookup tables. These components includes text labels and cursors to help users see the relation between numerical values and colors. The Display attached to map is updated when the user changes the color map. One type of constructor lets the application define the range of values mapped to color, the other uses the range of values defined by auto-scaling (if not range has been set yet by auto-scaling, the component listens for the appropriate event).

Users control LabeledRGBWidget and LabeledRGBAWidget pseudo color tables using the mouse. Click the left mouse button in the top part of the widget and drag to redraw the either the red, green, blue or alpha color graph. Click the center or right button to switch between red, green, blue and alpha graphs. Click the left mouse button in the bottom part of the widget and drag the arrow to see which RealType values are associated with the colors in the color bar.

```
/** this will be labeled with the name of map's RealType;
    the range of RealType values mapped to color is taken from
    map.getRange() - this allows a color widget to be used with
    a range of values defined by auto-scaling from displayed Data;
    if map's range values are not available at the time this
    constructor is invoked, the LabeledRGBWidget becomes a
    ScalarMapListener and sets its range when map's range is set;
    the DisplayRealType of map must be Display.RGB and should
    already be added to a Display */
public LabeledRGBWidget(ScalarMap map)
       throws VisADException, RemoteException;

/** this will be labeled with the name of map's RealType;
    the range of RealType values (min, max) is mapped to color
    as defined by an interactive color widget;
    the DisplayRealType of map must be Display.RGB and should
    already be added to a Display */
public LabeledRGBWidget(ScalarMap map, float min, float max)
       throws VisADException, RemoteException;

/** this will be labeled with the name of map's RealType;
    the range of RealType values (min, max) is mapped to color
    as defined by an interactive color widget; table initializes
```

```
    the color lookup table, organized as float[TABLE_SIZE][3]
    with values between 0.0f and 1.0f;
    the DisplayRealType of map must be Display.RGB and should
    already be added to a Display */
public LabeledRGBWidget(ScalarMap map, float min, float max,
                        float[][] table)
       throws VisADException, RemoteException;

/** this will be labeled with the name of map's RealType;
    the range of RealType values mapped to color is taken from
    map.getRange() - this allows a color widget to be used with
    a range of values defined by auto-scaling from displayed Data;
    if map's range values are not available at the time this
    constructor is invoked, the LabeledRGBAWidget becomes a
    ScalarMapListener and sets its range when map's range is set;
    the DisplayRealType of map must be Display.RGBA and should
    already be added to a Display */
public LabeledRGBAWidget(ScalarMap map)
       throws VisADException, RemoteException;

/** this will be labeled with the name of map's RealType;
    the range of RealType values (min, max) is mapped to color
    as defined by an interactive color widget;
    the DisplayRealType of map must be Display.RGBA and should
    already be added to a Display */
public LabeledRGBAWidget(ScalarMap map, float min, float max)
       throws VisADException, RemoteException;

/** this will be labeled with the name of map's RealType;
    the range of RealType values (min, max) is mapped to color
    as defined by an interactive color widget; table initializes
    the color lookup table, organized as float[TABLE_SIZE][4]
    with values between 0.0f and 1.0f;
    the DisplayRealType of map must be Display.RGBA and should
    already be added to a Display */
public LabeledRGBAWidget(ScalarMap map, float min, float max,
                         float[][] table)
       throws VisADException, RemoteException;
```

8.1.3 LabeledRGBWidget and LabeledRGBAWidget Methods

Generally useful methods of LabeledRGBWidget and LabeledRGBAWidget include:

```
/** set maximum size of widget using java.awt.Dimension */
public setMaximumSize(Dimension d);
```

8.1.4 SelectRangeWidget Constructor

The SelectRangeWidget class extends java.awt.Canvas. It provides a way for users to interactively change Display.SelectRange bounds. This component includes text labels and cursors to help users see and control the range of selected values. The Display attached to the ScalarMap constructor is updated when the user changes the range. One constructor lets the application define the range of selectable range values, the other uses the range of values defined by auto-scaling (if not range has been set yet by auto-scaling, the component listens for the appropriate event).

Users control the SelectRangeWidget range using the mouse. Change either end of the range by dragging with the mouse. Click in the middle of the range to move both ends of the range in unison.

```
/** this will be labeled with the name of map's RealType;
    the range of RealType values defining the bounds of the
    selectable range is taken from map.getRange() - this allows
    a SelectRangeWidget to be used with a range of values defined
    by auto-scaling from displayed Data; if map's range values
    are not available at the time this constructor is invoked,
    the SelectRangeWidget becomes a ScalarMapListener and sets
    its range when map's range is set;
    the DisplayRealType of map must be Display.SelectRange and
    should already be added to a Display */
public SelectRangeWidget(ScalarMap map)
       throws VisADException, RemoteException;

/** this will be labeled with the name of map's RealType;
    the range of RealType values (min, max) is defines the
    bounds of the selectable range;
    the DisplayRealType of map must be Display.SelectRange and
    should already be added to a Display */
public SelectRangeWidget(ScalarMap map, float min, float max)
       throws VisADException, RemoteException;
```

8.1.5 AnimationWidget Constructor

The AnimationWidget class extends JPanel. It provides a way for users to interactively change parameters of AnimationControl. The Display attached to the ScalarMap constructor argument is updated when the user changes animation parameters.

```
/** the DisplayRealType of map must be Display.Animation and
    should already be added to a Display */
public AnimationWidget(ScalarMap map)
       throws VisADException, RemoteException;

/** the DisplayRealType of map must be Display.Animation and
    should already be added to a Display; st is the dwell time
    per animation step, in milliseconds */
public AnimationWidget(ScalarMap map, int st)
       throws VisADException, RemoteException;
```

8.1.6 ContourWidget Constructor

The ContourWidget class extends JPanel. It provides a way for users to interactively change parameters of ContourControl. The Display attached to the ScalarMap constructor argument is updated when the user changes animation parameters.

```
/** the DisplayRealType of map must be Display.IsoContour and
    should already be added to a Display */
public ContourWidget(ScalarMap map)
       throws VisADException, RemoteException;

/** the DisplayRealType of map must be Display.IsoContour and
    should already be added to a Display; surf is an initial
    iso-surface value */
public ContourWidget(ScalarMap map, float surf)
```

```
        throws VisADException, RemoteException;

  /** the DisplayRealType of map must be Display.IsoContour and
      should already be added to a Display; initial iso-lines are
      generated for values in an arithmetic progression centered at
      base with in increment of interval, between min and max */
  public ContourWidget(ScalarMap map, float interval, float min,
                       float max, float base)
        throws VisADException, RemoteException;

  /** the DisplayRealType of map must be Display.IsoContour and
      should already be added to a Display; initial iso-lines are
      generated for values in an arithmetic progression centered at
      base with in increment of interval, between min and max; surf
      is an initial iso-surface value; if update is true, then the
      range of legals min and max values is updated whenever the
      range of data values in map is auto-scaled */
  public ContourWidget(ScalarMap map, float interval, float min,
                       float max, float base, float surf,
                       boolean update)
        throws VisADException, RemoteException;
```

8.1.7 GMCWidget Constructor

The GMCWidget class extends JPanel. It provides a way for users to interactively change parameters of GraphicsModeControl. The Display attached to the GraphicsModeControl constructor argument is updated when the user changes parameters.

```
  /** create a GMCWidget for control */
  public GMCWidget(GraphicsModeControl control);
```

9. Simplified Use of VisAD

There is no doubt that VisAD is powerful, but that power implies a breadth of choices which can be confusing to developers. This section is about some classes and methods that are intended to make VisAD easier to use.

The simplest way to use VisAD is through its Spread Sheet, which is described in Section 10 and provides a way to use VisAD without any user programming at all. It enables users to read a variety of file formats, display their contents, change the way they are displayed, and perform simple arithmetic operations on them (e.g., subtract two images and display the result).

The visad.util.DataUtility class provides the simplest level of support for users who need to write their own programs. For example, if you have a program that computes some image pixel brightnesses and want to display them, you can use the static *makeImage* and *makeSimpleDisplay* methods of DataUtility to do that in just a couple lines of code. Given an array of pixel brightnesses:

```
  float pixels[nlines][nelements]
```

the following code creates a VisAD image and displays it:

```
  FlatField image = DataUtility.makeImage(pixels);
```

```
DisplayImpl display = DataUtility.makeSimpleDisplay(image);

JFrame jframe = new JFrame("simple image display");
jframe.setContentPane(((JPanel) display.getComponent());
jframe.pack();
jframe.setVisible(true);
```

Only the first two lines of code have to do with VisAD: the first creates the data object (with class FlatField) and the second displays it. The other four lines of code set up a minimal Java frame for the display. The *makeSimpleDisplay* method of DataUtility can create a simple display for almost any VisAD data object. We will add more methods to DataUtility similar to *makeImage*, to construct other simple kinds of data objects, like 2-D and 3-D grids, and tables.

The visad.MathType class provides some useful methods for users who need to explicitly construct and manipulate MathTypes for more complex data objects. Its *stringToType* method enables users to construct complex MathTypes from the shorthand notation described in Section 3.1. Recall that the shorthand MathType for multi-spectral satellite image of Earth is:

```
( (latitude, longitude) ->
  (radiance_channel_1, ..., radiance_channel_N) )
```

The shorthand MathType for the output of a weather model is:

```
( time -> ( (latitude, longitude, altitude) ->
  (temperature, pressure, dewpoint, wind_u, wind_v, wind_w) ) )
```

And the shorthand MathType for a set of map boundaries is:

```
set ( (latitude, longitude) )
```

The static *stringToType* method of MathType takes a String argument, which is assumed to be in this shorthand notation, and returns the corresponding MathType (of course, MathTypes returned by *stringToType* do not include any non-null default Units, CoordinateSystems or Sets). The *prettyString* method of MathType returns a String with this shorthand notation for any VisAD MathType.

The *guessMaps* method of MathType returns an array of ScalarMaps appropriate for displaying data objects with this MathType. The *guessMaps* method has one argument, a boolean *threeD*, which is true if a 3-D display is OK.

10. The VisAD Spread Sheet

The visad.ss package is a "generic" spreadsheet user interface for VisAD. It is intended to be poweful and flexible, and it can be used to visualize many types of data, without any programming. It supports many features of a traditional spreadsheet, such as formulas. The package also provides a class structure such that developers can easily create their own user interfaces using Spread Sheet cells from the visad.ss package.

For up-to-date information about the VisAD Spread Sheet, see the VisAD Spread Sheet web page at http://www.ssec.wisc.edu/~curtis/ss.html (this is linked of of the main VisAD web page at http://www.ssec.wisc.edu/~billh/visad.html).

10.1 Spread Sheet Classes

The VisAD Spread Sheet consists a number of classes, plus the following gif files as user interface icons: cancel.gif, copy.gif, cut.gif, display.gif, import.gif, mappings.gif, ok.gif, open.gif, paste.gif, save.gif, show.gif. The Spread Sheet classes are:

BasicSSCell

This class can be instantiated and added to a JFC user interface. It represents a single spreadsheet cell with some basic capabilities. It is designed to be "quiet" (i.e., it throws exceptions rather than displaying errors in error message dialog boxes).

FancySSCell

This class is an extension of BasicSSCell that can be instantiated and added to a JFC user interface to provide all of the capabilities of a BasicSSCell, plus some additional, "fancy" capabilities. It is designed to be "loud" (i.e., it displays errors in error message dialog boxes rather than throwing exceptions).

Formula

This class converts formulas to postfix notation for evaluation on a stack. It is used by FormulaCell.

FormulaCell

This class is used internally by BasicSSCell to evaluate formulas.

MappingDialog

This class is a dialog box allowing the user to specify ScalarMaps for the current data set.

SpreaSheet

This is the main Spread Sheet user interface class. It manages multiple FancySSCells.

SSLayout

This is the layout manager for the spreadsheet cells and their labels.

10.2 Features of the SpreadSheet User Interface

10.2.1 Basic Commands

The spreadsheet cell with the yellow border is the current, highlighted cell. Any operation you perform (such as importing a data set), will affect the highlighted cell. To change which cell is highlighted, click inside the desired cell with a mouse button, or press the arrow keys. You can also resize the spreadsheet cells, to allow some cells to be larger than others, by dragging the yellow block between cell labels.

10.2.2 Menu Commands

10.2.2.1 File Menu

Here are the commands from the File menu:

Import data

Brings up a dialog box that allows the user to select a file for the Spread Sheet to import to the current cell. Currently, VisAD supports the following file types:
GIF, JPEG, netCDF, HDF-EOS, FITS, Vis5D, and serialized data.

Note: You must have the HDF-EOS and Vis5D file adapter native C code compiled in order to import data sets of those types. See the Spread Sheet web page for information on how to compile this native code.

Export data to netCDF

Exports the current cell to a file in netCDF format. A dialog box will appear to let you select the name and location of the netCDF file. If the file exists, it will be overwritten.

Export serialized data

Exports the current cell to a file in serialized data format (the "VisAD"form). A dialog box will appear to let you select the name and location of theserialized data file. If the file exists, it will be overwritten.

WARNING: Exporting a cell as serialized data is a handy and portable way to store data, but each time the VisAD Data class hierarchy changes, old serialized data files become obsolete and will no longer load properly. For long term storage of your data, use the "Export data to netCDF" command.

Exit

Quits the VisAD SpreadSheet User Interface.

10.2.2.2 Edit Menu

Here are the commands from the Edit menu:

Cut

Moves the current cell to the clipboard.

Copy

Copies the current cell to the clipboard.

Paste

Copies the cell in the clipboard to the current cell.

Clear

Clears the current cell.

10.2.2.3 Setup Menu

Here are the commands from the Setup menu:

New

Clears all spreadsheet cells; starts from scratch.

Open

Opens a "spreadsheet file." Spreadsheet files are small, containing only the instructions needed to recreate a spreadsheet. They do not contain any actual data, but rather the file names and formulas of the cells.

Save

Saves a "spreadsheet file" under the current name.

Save as

Saves a "spreadsheet file" under a new name.

10.2.2.4 Display Menu

Here are the commands from the Display menu:

Edit Mappings

Brings up a dialog box which lets you change how the Data object is mapped to the Display. Click a RealType object on the left (or from the MathType display at the top), then click a display icon from the display panel in the center of the dialog. The "Current Mappings" box on the lower right will change to reflect which mappings you've currently set up. When you've set up all the mappings to your liking, click the Done button and the Spread Sheet will try to display the data object. To close the dialog box without applying any of the changes you made to the mappings, click the Cancel button. You can also highlight items from the "Current Mappings" box, then click "Clear selected" to remove those mappings from the list, or click "Clear all" to clear all mappings from the list and start from scratch.

3-D (Java3D)

Sets the current cell's display dimension to 3-D. This setting requires Java3D. If you do not have Java3D installed, this option will be grayed out.

2-D (Java2D)

Sets the current cell's display dimension to 2-D. This uses Java2D, which is included with the Java 1.2. However, in this mode, nothing can be mapped to ZAxis, Latitude, or Alpha. For computers without 3-D acceleration, this mode will provide better performance,

but the display quality will not be as good as 2-D (Java3D). If you do not have Java3D installed, this is the only available mode.

2-D (Java3D)

Sets the current cell's display dimension to 2-D. This requires Java3D. In this mode, nothing can be mapped to ZAxis or Latitude (but things can be mapped to Alpha). On computers with 3-D acceleration, this mode will probably provide better performance than 2-D (Java2D). It also has better display quality than 2-D (Java2D). If you do not have Java3D installed, this option will be grayed out.

10.2.2.5 Options Menu

Here are the commands from the Options menu:

Auto-switch to 3-D

If this option is checked, cells will automatically switch to 3-D display mode when mappings are used that require 3-D display mode. In addition, it will switch to mode 2-D (Java3D) from mode 2-D (Java2D) if anything is mapped to Alpha or RGBA. If you do not have Java3D installed, this option is grayed out. Otherwise, this option is checked by default.

Auto-detect mappings

If this option is checked, the Spread Sheet will attempt to detect a good set of mappings for a newly loaded data set and automatically apply them. This option is checked by default.

Show formula evaluation errors

If this option is checked, dialog boxes will pop up explaining why any formulas entered are illegal or could not be evaluated. If this option is not checked, the only notification of an error is a large X through the current cell.

Show VisAD controls

Displays the set of controls relevant to the current cell (these controls are displayed by default, but could become hidden at a later time). This option is not a checkbox, but rather just redisplays the VisAD Controls for the current cell if they have been closed by the user.

10.2.3 Toolbars

10.2.3.1 Main Toolbar

The main toolbar provides shortcuts to the following menu items: File Import, Edit Cut, Edit Copy, Edit Paste, Display Edit Mappings, and Options Show VisAD Controls. The main toolbar has tool tips so each button can be easily identified.

10.2.3.2 Formula Toolbar

10.2.3.2.1 Description

The formula toolbar is used for entering file names, URLs, and formulas for the current cell. If you enter the name of a file in the formula text box, the Spread Sheet will attempt to import the data from that file. If you enter a URL, the Spread Sheet will try to download and import the data from that URL. If you enter a formula, it will attempt to parse and evaluate that formula. If a formula entered is invalid for some reason, the answer cannot be computed, or the file entered does not exist, the cell will have a large X through it instead of the normal data box. If the data box appears, the cell was computed successfully and mappings can be set up.

10.2.3.2.2 How To Enter Formulas

To reference cells, keep in mind that each column is a letter (the first column is 'A', the second is 'B', and so on), and each row is a number (the first row is '1', the second is '2', and so on). So, the cell on the top-left is A1, the cell on A1's right is B1, and the cell directly below A1 is A2, etc.

Any of the following can be used in formula construction:

1) Formulas can use any of the basic operators:
+ add, - subtract, * multiply, / divide, % remainder, ^ power

2) Formulas can use any of the following binary functions:
MAX, MIN, ATAN2, ATAN2DEGREES

3) Formulas can use any of the following unary functions:
ABS, ACOS, ACOSDEGREES, ASIN, ASINDEGREES, ATAN, ATANDEGREES, CEIL, COS, COSDEGREES, EXP, FLOOR, LOG, RINT, ROUND, SIN, SINDEGREES, SQRT, TAN, TANDEGREES, NEGATE

4) Unary minus syntax (e.g., B2 * -A1) is supported.

5) Derivatives are supported with the syntax:
d(DATA)/d(TYPE)
where DATA is a Function, and TYPE is the name of a RealType present in the Function's domain. This syntax calls Function's derivative() method with an error_type of Data.NO_ERRORS.

6) Function evaluation is supported with the syntax:
DATA1(DATA2)
where DATA1 is a Function and DATA2 is a Real or a RealTuple. This syntax calls Function's evaluate() method.

7) You can obtain an individual sample from a Field with the syntax:
DATA(N)

where DATA is the Field, and N is a literal integer. Use DATA(0) for the first sample of DATA. This syntax calls Field's getSample() method.

8) You can obtain one component of a Tuple with the syntax:

DATA.N

where DATA is a Tuple and N is a literal integer. Use DATA.0 for the first Tuple component of DATA. This syntax calls Tuple's getComponent() method.

9) You can extract part of a field with the syntax:

EXTRACT(DATA, N)

where DATA is a Field and N is a literal integer. This syntax calls Field's extract() method.

10) Formulas are not case sensitive.

Some examples of valid formulas for cell A1 are:

SQRT(A2 + B2^5 - MIN(B1, -C1))
d(A2 + B2)/d(ImageElement)
A2(A3)
C2.6
(B1 * C1)(A3).1

Once you've typed in a formula, press Enter or click the green check box button to the left of the formula entry text box to apply the formula. The red X button will cancel your entry, restoring the formula to its previous state. The open folder button to the right of the formula entry text box is a shortcut to the File menu's Import Data menu item.

10.3. Future Plans

Here's what's coming in the future for the VisAD Spread Sheet:

1. Spreadsheet column and row addition and deletion
2. Multiple data per cell
3. Direct manipulation support
4. Distributed Cells, Data, etc.
5. Remote Spread Sheet cloning with collaboration
6. Formula enhancements, including composition of multiple Data objects (such as creating an animation from multiple spreadsheet cells), and dynamic linkage of Java code into formulas
7. Misc. user interface enhancements
8. And of course, bug fixes

11. Extending the VisAD Java Class Library

Object-oriented programming languages like Java allow classes to be extended, and we have tried to capitalize on this in the design of VisAD. We have specifically designed classes to be extensible to allow users to add needed functionality. For example:

1. The Set class may be extended to define new Field sampling topologies or new algorithms for interpolating between samples.
2. The CoordinateSystem class may be extended to define new coordinate transformation algorithms.
3. The Function class may be extended to define non-sampled approximations to functions, such as harmonic series.
4. The FlatField class may be extended to define specialized classes for images, grids, tables, etc.
5. The Real class may be extended to define high-precision, multi-word approximations to real numbers.
6. The Data class and its subclasses may be extended to import new file formats (or other data sources) as VisAD Data objects.
7. The DataRenderer, DisplayRenderer, DisplayRealType, Control and ShadowType classes may be extended to define new basic rendering techniques including new direct manipulation techniques.
8. The DataRenderer, DisplayRenderer, Control and ShadowType classes may be extended to provide visualization support based on graphics APIs other than Java3D and Java2D.
9. The Cell class may be extended to define new computational algorithms.
10. JavaBean components can be defined that encapsulate the Data, Display, Cell and user interface (e.g., VisADSlider) classes to provide a visual programming metaphor for building VisAD applications. We plan to develop such JavaBean components for VisAD, and encouage others to do so.

The ways that classes can be extended are described in more detail in the sections that document specific class constructors and methods. We recommend that extensions be put into separate packages. We will be very happy to provide links from the VisAD web page to web pages describing and serving VisAD extension packages. Please send an email message to Bill Hibbard at hibbard@facstaff.wisc.edu if you develop a VisAD extension package.

12. Application Examples

The easiest way to develop new VisAD applications is by following the pattern of existing applications. Thus we provide the following source code examples for typical visualization, analysis and collaboration operations.

12.1 The DisplayTest Class

The DisplayTest and associated TestNN classes in the visad/examples directory (note these classes do not include a package statement) implement many small tests of the VisAD system's visualization and interaction techniques. They are an excellent source of VisAD coding examples. To see a list of tests, change to the visad/examples directory and enter the command:

```
java DisplayTest
```

12.2 Visualizing the HSV Color CoordinateSystem

The HSVDisplay application in the visad/examples directory provides interactive exploration of the relation between the HSV and RGB color spaces. Here is a section of code from HSVDisplay that illustrates how CoordinateSystems can be used implicitly in Display ScalarMaps:

```
// define an rgb color space
// (not to be confused with system's RGB DisplayTupleType)
RealType red = new RealType("red", null, null);
RealType green = new RealType("green", null, null);
RealType blue = new RealType("blue", null, null);
RealTupleType rgb = new RealTupleType(red, green, blue);

// define an hsv color space
// (not to be confused with system's HSV DisplayTupleType)
RealType hue = new RealType("hue", CommonUnit.degree, null);
RealType saturation = new RealType("saturation", null, null);
RealType value = new RealType("value", null, null);
// define the relation between the hsv and rgb color spaces
// using the same HSVCoordinateSystem that the system uses to
// define the relation between its RGB and HSV color spaces
CoordinateSystem hsv_system = new HSVCoordinateSystem(rgb);
RealTupleType hsv = new RealTupleType(hue, saturation, value,
                                      hsv_system, null);

// construct a sampling of the hsv color space;
// since hue is composed of six linear (in rgb) pieces with
// discontinuous derivative bwteen pieces, it should be sampled
// at 6*n+1 points with n not too small;
// for a given hue, saturation and value are both linear in rgb
// so 2 samples suffice for each of them;
// the HSV - RGB transform is degenerate at satruration = 0.0
// and value = 0.0 so avoid those values;
// hue is in Units of degrees so that must be used in the Set
// constructor
Linear3DSet cube_set =
  new Linear3DSet(hsv, 0.0, 360.0, 37,
                       0.01, 1.0, 2,
                       0.01, 1.0, 2, null,
                  new Unit[] {CommonUnit.degree, null, null},
                  null);

// construct a DataReference to cube_set so it can be displayed
DataReference cube_ref = new DataReferenceImpl("cube");
cube_ref.setData(cube);

// skip some code to set up UI . . .

// construct a Display
DisplayImplJ3D display1 = new DisplayImplJ3D("display1");

// map rgb to the Display spatial coordinates;
// note that red, green and blue do not occur in cube_set
// but are related to hue, saturation and reference via a
// CoordinateSystem that will be applied implicitly by
// Display logic
display1.addMap(new ScalarMap(red, Display.XAxis));
display1.addMap(new ScalarMap(green, Display.YAxis));
display1.addMap(new ScalarMap(blue, Display.ZAxis));
```

```
// define colors for points in hsv space
display1.addMap(new ScalarMap(hue, Display.Hue));
display1.addMap(new ScalarMap(saturation, Display.Saturation));
display1.addMap(new ScalarMap(value, Display.Value));

// construct mappings for interactive iso-surfaces of
// hue, saturation and value;
// the ContourControls must be extracted from these ScalarMaps
// to support interactive control of iso-surface levels
ScalarMap maphcontour = new ScalarMap(hue, Display.IsoContour);
display1.addMap(maphcontour);
ContourControl controlhcontour =
  (ContourControl) maphcontour.getControl();

ScalarMap mapscontour = new ScalarMap(saturation, Display.IsoContour);
display1.addMap(mapscontour);
ContourControl controlscontour =
  (ContourControl) mapscontour.getControl();

ScalarMap mapvcontour = new ScalarMap(value, Display.IsoContour);
display1.addMap(mapvcontour);
ContourControl controlvcontour =
  (ContourControl) mapvcontour.getControl();

// display cube_set;
// it will be dispayed as a set of colored, interactive hue,
// saturation and value iso-surfaces, transformed into rgb space
display1.addReference(cube_ref);
```

The HSVCoordinateSystem class is used internally by the system for Displays that include ScalarMaps to Display.Hue, Display.Saturation, Display.Value or Display.HSV. Internally, it always has Reference Display.DisplayRGBTuple. However, the HSVDisplay application constructs a HSVCoordinateSystem whose Reference it maps to Display spatial axes in order to spatially visualize the geometry of the relation between HSV and RGB color spaces.

12.3 Collaborative GOES Satellite Sounding Analysis

The GoesCollaboration application is an interactive and collaborative spread sheet for experimenting with algorithms for analyzing multi-spectral GOES satellite data, adapted from an application written by Paolo Antonelli and Bob Aune under VisAD version 1.1 (the C implementation). Figure 1 (which is supplied with some hard copies of this guide, and is also available at http://www.ssec.wisc.edu/~billh/figure1.gif) is a screen shot of this application, showing its four Displays and four slider widgets (there are five Cells linking the Displays and sliders computationally). The lower-left Display shows vertical atmospheric profiles of pressure, temperature, water vapor and ozone. When users re-draw these profiles (this is an example of direct manipulation), the underlying data objects change, which triggers Cells to re-compute the data objects shown in the other Displays. The four slider widgets on the left can also be used to change simple Real data values, which trigger other Cells to re-compute more complex data values.

The GoesCollaboration application is part of the visad.paoloa package. Its source code, data files and installation instructions are available from the VisAD web page at:

```
http://www.ssec.wisc.edu/~billh/visad.html
```

Once the GoesCollaboration application is running on one machine, it may be started on other machines and connected to the first. This is specified by typing the IP name of the first machine as the command line argument of GoesCollaboration on other machines. For example, we could start the first (server) copy of GoesCollaboration on sparc.ssec.wisc.edu by typing the commands:

```
rmiregistry &
java visad.paoloa.GoesCollaboration
```

Then we could start GoesCollaboration (client) on any number of other machines by typing:

```
java visad.paoloa.GoesCollaboration sparc.ssec.wisc.edu
```

These copies of GoesCollaboration will all be connected together so that when the user drags sliders or re-draws atmospheric profiles in one copy of GoesCollaboration, all the users will see these changes and their computational consequences in their copies of GoesCollaboration.

The complete and annotated source code for the GoesCollaboration application is listed in Appendix B. Note that GoesCollaboration initially determines whether it is started as a server (with no argument) or as a client (with the IP name of the server as an argument). As a server it constructs a set of Data and DataReference objects which it serves via a RemoteServerImpl object bound to a URL. It also constructs sets of Cell, Display and VisADJSlider (user interface) objects connected to the DataReference objects. As a client it obtains references to RemoteDataReference objects from the server, then constructs Display and VisADJSlider objects which it connects to the RemoteDataReference objects from the server.

The GoesCollaboration application includes Fortran implementations of its science algorithms, which are invoked via JNI through C wrappers. These are only invoked by the applications computational Cells, and hence are only invoked by the server copy of GoesCollaboration. It is possible to run client copies of GoesCollaboration on machines that cannot run the Fortran science algorithms.

12.4 A Steerable Shallow Fluid Model

The ShallowFluid application allows users to interactively steer Bob Aune's 2-D shallow fluid model. Users can experiment with changes to the gravity constant and other physical parameters and see their affect on fluid flow. Users can also experiment with numerical parameters. In particular, users may increase the number of seconds between simulated time steps and visualize the development of numerical instability.

The ShallowFluid application is part of the visad.aune package. Its source code, data files and installation instructions are available from the VisAD web page at:

```
http://www.ssec.wisc.edu/~billh/visad.html
```

12.5 The JMet Weather Simulation Visualizer

The JMet system is distributed with VisAD in the visad.jmet package and its initial release can be used to visualize weather model output in netCDF files. For more information, see the JMet web page at:

```
http://allegro.ssec.wisc.edu/jmet/
```

12.6 Image Animation Using Java2D

The SimpleAnimate class in the visad/examples directory provides an example of animating a time sequence of satellite images using Java2D. In order to run the SimpleAnimate application you need to download and uncompress the netCDF file "images.nc" from:

```
ftp://iris.ssec.wisc.edu/pub/visad-2.0/images.nc.Z
```

To run this application, change to the visad/examples directory and enter the command:

```
java SimpleAnimate (step_time_in_ms)
```

where (step_time_in_ms) is an optional paramter giving the animation step time in milliseconds. The default value is 1000 ms.

12.7 Earth Topography and Bathymetry

The Earth class in the visad/examples directory generates a nice looking interactive 3-D Earth globe with topography and bathymetry. In order to run the Earth application you need to download and uncompress the netCDF file "lowresTerrain.nc" from:

```
ftp://iris.ssec.wisc.edu/pub/visad-2.0/lowresTerrain.nc
```

To run this application, change to the visad/examples directory and enter the command:

```
java -Xmx64m Earth lowresTerrain.nc
```

13. Caveats and Future Plans

We wrote the VisAD Java class library because we believe that Java will become the universal programming language supporting distributed object programming across the Internet, and because we have faith that compiler and chip designers will bring Java to the performance levels of other languages (this faith has been justified by the Java 2 Solaris Production Release, which is as fast as C).

However, for the initial release of VisAD, ubiquity and performance are problems. As described in Section 3.9, VisAD makes few method calls, so that its speed should be good once compilers are able to get good speed on loops over arrays of floats and doubles. Memory performance is also a problem with the current release of Java3D, which is used by VisAD. This should be improved in version 1.2 of Java3D. Note that VisAD Data objects can be stored efficiently with appropriate choices of range Sets in FlatField constructors, although many file adapters just store all input values as (4-byte) floats. If VisAD throws

an OutOfMemoryException, increase memory size of the Java interpreter with the -mx command line option.

The AuditTrail class is not implemented in the initial release of VisAD. Display logic is unimplemented for some combinations of ScalarMaps and MathTypes. The ErrorEstimate, ProductSet and UnionSet classes are not well tested. The file format adapters may fail to adapt some files, and may not adapt all information in other files.

The initial release of VisAD was essentially simultaneous with the public early access release of Java3D. Java 2 (aka JDK 1.2) is are required for VisAD. Either Java2D (included in Java 2) or Java3D can be used for displays. Most vendors have Java 2 alpha releases as of May 1999.

We will continue to add file format adapters and associated metadata classes such as new CoordinateSystems and Sets. We will develop packages for statistics and mathematical analysis operations. We will add more support for collaborative user interfaces, and will develop a number of generic user interfaces such as a general spread sheet.

We will try to support developers using VisAD, by fixing bugs, answering questions and adding requested features. VisAD's extensibility should enable developers to add new features to the system.

13.1 JavaBean Components

Clearly, since a VisAD consists of a linked network of Data, Display, user interface and computastion Cell objects, users should be able to build these networks visually using JavaBean components. We plan to implement a variety of JavaBeans to help users build networks of VisAD objects.

14. For More Information and Help with Problems

For more information, please see the VisAD web page at:

```
http://www.ssec.wisc.edu/~billh/visad.html
```

You can get help with problems from the VisAD mailing list at:

```
visad-list@ssec.wisc.edu
```

You can join the VisAD mailing list by sending an email message to:

```
majordomo@ssec.wisc.edu
```

that includes the line:

```
subscribe visad-list
```

in the body of the message (not the subject line). If you get an Exception in a VisAD class that you need our help with, it will be very useful if you paste the text of the stack trace from the Exception into your email message. In many cases, Exceptions from VisAD will tell you that you have passed illegal arguments to a method of a VisAD class. Hopefully in such cases our Exception messages will help you find errors in your application.

If you are interested in collaborating with us on VisAD developments please send email to Bill Hibbard at hibbard@facstaff.wisc.edu.

15. References

1. Baltuch, M. S., 1997; Unidata's Internet data distribution (IDD) system: two years of data delivery. Proc, 13th Int. Conf. on Interactive Information and Processing for Meteorology, Oceanography amd Hydrology, Amer. Meteor. Soc., 168-171.

2. Beshers, C., and S. Feiner, 1992; Automated design of virtual worlds for visualizing multivariate relations. Proc. Visualization '92, IEEE. 283-290.

3. Haber, R. B., B. Lucas and N. Collins, 1991; A data model for scientific visualization with provisions for regular and irregular grids. Proc. Visualization 91. IEEE. 298-305.

4. Hasler, A. F., J. Dodge, and R. H. Woodward, 1991; A High Performance Interactive Image Spreadsheet. Preprints of the Seventh International Conference on Interactive Information and Processing systems for Meteorology, Oceanography and Hydrology, New Orleans, Amer. Meteor. Soc., 187-194.

5. Hibbard, W., 1986; 4-D display of meteorological data. Proceedings, 1986 Workshop on Interactive 3D Graphics. Chapel Hill, ACM Siggraph, 23-36.

6. Hibbard, W., and D. Santek, 1990; The Vis5D system for easy interactive visualization. Proc. Visualization '90, San Francisco, IEEE. 28-35.

7. Hibbard, W., D. Santek, and G. Tripoli, 1991; Interactive atmospheric data access via high speed networks. Computer Networks and ISDN Systems, 22, 103-109.

8. Hibbard, W., C. Dyer and B. Paul, 1992; Display of scientific data structures for algorithm visualization. Proc. Visualization '92, Boston, IEEE, 139-146.

9. Hibbard, W. L., B. E. Paul, D. A. Santek, C. R. Dyer, A. L. Battaiola, and M-F. Voidrot-Martinez, 1994; Interactive visualization of Earth and space science computations. IEEE Computer 27(7), 65-72.

10. Hibbard, W, J. Anderson, I. Foster, B. Paul, R. Jacob, C. Schafer, and M. Tyree, 1996; Exploring Coupled Atmosphere-Ocean Models Using Vis5D. Int. J. of Supercomputer Applications, 10(2), 211-222.

Appendix A Constraints on ScalarMaps and MathTypes

In order to describe the constraints on ScalarMaps and MathTypes used by the DefaultDisplayRendererJ3D and DefaultDataRendererJ3D classes we must first define a few terms.

A TupleType is *flat* if all its components are RealTypes or RealTupleTypes. A FunctionType is *flat* if its range is a RealType or a flat TupleType (note that a flat FunctionType is appropriate for the MathType of a FlatField).

The MathType of each displayed Data object defines a tree structure whose leaves are RealTypes (TextTypes are ignored). We define *terminal* nodes as nodes in this tree that are:

1. Flat FunctionTypes.
2. SetTypes.
3. Flat TupleTypes that are not part of terminal FunctionTypes or other terminal TupleTypes.
4. If a displayed Data object is a Real, then its RealType is terminal.

Each terminal node in the MathType tree defines a *path* through containing TupleTypes and FunctionsTypes back to the root of the tree. RealTypes *occur* in this path if they are part of:

1. The terminal node of the path.
2. A FunctionType in the path, as components of its domain RealTupleType.
3. A TupleType in the path, either as a RealType component or a RealType sub-component of a RealTupleType component.

Now the constraints on ScalarMaps and MathTypes can be described as follows:

1. No two ScalarMaps may have the same RealType and DisplayRealType (i.e., two ScalarMaps may not be identical), unless the DisplayRealType is Display.Shape.
2. A RealType mapped to Animation or SelectValue may only occur in the MathType of a displayed Data object as the 1-D domain of a FunctionType.
3. Only one RealType occurring in a path to a terminal node may be mapped to Animation.
4. No RealType may occur more than once in a path, unless that RealType is not mapped to any DisplayRealType.
5. None of the DisplayRealTypes declared as *Single* may mapped from multiple RealTypes occurring in a path. Single DisplayRealTypes are: XAxis, YAxis, ZAxis, Latitude, Longitude, Radius, Animation, ShapeScale, Text, Flow1X, Flow1Y, Flow1Z, Flow2X, Flow2Y and Flow2Z.
6. RealTypes occurring in a path may not be mapped to components of multiple display spatial tuples. These are DisplaySpatialCartesianTuple and any DisplayTupleTypes with a CoordinateSystem whose Reference is DisplaySpatialCartesianTuple (e.g., DisplaySpatialSphericalTuple).

In addition to these constraints, there are many other combinations of ScalarMaps and MathTypes that are nonsensical, that produce uninteresting or trivial data depictions, or that are very difficult or ambiguous to render. Common sense is the best rule of thumb for defining ScalarMaps.

Appendix B The GoesCollaboration Application Source Code

```
//
// GoesCollaboration.java
//

package visad.paoloa;

// VisAD packages
import visad.*;
import visad.util.Delay;
import visad.util.VisADSlider;
import visad.java3d.DisplayImplJ3D;
import visad.java3d.TwoDDisplayRendererJ3D;
import visad.java3d.DirectManipulationRendererJ3D;


// Java packages
import java.io.File;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import java.rmi.AccessException;
import java.rmi.Naming;
import java.net.MalformedURLException;

// JFC packages
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.border.*;

// AWT packages
import java.awt.*;
import java.awt.event.*;

/**
   GoesCollaboration implements the interactive and collaborative
   Goes satellite sounding retrieval application using VisAD 2.0.
   It is rewritten from the IRGS.v application developed for
   VisAD 1.1 by Paolo Antonelli.<P>
*/
public class GoesCollaboration extends Object {

  /** RemoteServerImpl for server
      this GoesCollaboration is a server if server_server != null */
  RemoteServerImpl server_server;

  /** RemoteServer for client
      this GoesCollaboration is a client if client_server != null */
  RemoteServer client_server;

  /** declare MathTypes */
  RealType nchan;
  RealType indx;
  RealType nl;
  RealType tbc;
  RealType tbc_d;
  RealType wfn;
```

```
RealType pres;
RealType temp;
RealType mixr;
RealType ozone;
RealType pressure;
RealType data_real;
RealType diff;

/** declare DataReferences */
DataReference wfna_ref;
DataReference tempa_ref;
DataReference mixra_ref;
DataReference ozonea_ref;
DataReference presa_ref;
DataReference diff_col_ref;
DataReference diff_ref;
DataReference zero_line_ref;
DataReference smr_ref;
DataReference real_tbc_ref;
DataReference wfnb_ref;
DataReference wfna_old_ref;

/** slider DataReferences */
DataReference gzen_ref;
DataReference tskin_ref;
DataReference in_dx_ref;

/** the width and height of the UI frame */
public static int WIDTH = 1200;
public static int HEIGHT = 1000;

/** type 'java visad.paoloa.GoesCollaboration' to run this application;
    the main thread just exits, since Display, Cell and JFC threads
    run the application */
public static void main(String args[])
       throws VisADException, RemoteException {
  // construct GoesCollaboration application
  GoesCollaboration goes = new GoesCollaboration(args);

  if (goes.client_server != null) {
    goes.setupClient();
  }
  else if (goes.server_server != null) {
    // load native method library (only needed for server)
    System.loadLibrary("GoesCollaboration");
    goes.setupServer();
  }
  else {
    // stand-alone (neither client nor server)
    // load native method library
    System.loadLibrary("GoesCollaboration");
    goes.setupServer();
  }
}

/**
   Construct the GoesCollaboration application, including Data
   objects, Display objects, Cell (computational) objects,
   and JFC (slider) user interface objects.  The Display,
   Cell and JFC objects include threads and links to Data
```

```
   objects (via DataReference objects).  Display and Cell
   threads wake up when linked Data objects change.  Display
   and JFC objects wake up on mouse events.  Display, Cell
   and JFC objects cause changes to Data objects.<P>

   Here's a summary of the event logic among Data, Displays,
   Cells, and JSliders:<P>

<PRE>
initialization ->
  zero_line = 0                                -> display4

slider <--> in_dx

slider <--> gzen

slider <--> tskin

slider <--> save_config

in_dx -> real_tbcCell
  real_tbc = re_read_1_c(in_dx)
  month = 6
  lat = real_tbc[18];
  (tempa, mixra, ozonea, presa) =
    get_profil_c(lat, month)                   -> display2

direct_manipualtion (in display2) ->
  (tempa, mixra, ozonea)                       -> display2

gzen, tskin, tempa, mixra, ozonea, presa -> wfnbCell
  wfnb = goesrte_2_c(gzen, tskin, tempa, mixra, ozonea, presa)

wfnb, real_tbc -> wfnaCell
  wfna = wfnb.wfn                              -> display1
  diff_DATA = wfnb.tbc[nl=1] - real_tbc        -> display4
  smr = RootMeanSquare(diff_DATA)              -> display4

save_config -> wfna_oldCell
  wfna_old = wfna

wfna, wfna_old -> diff_colCell
  diff_col = wfna - wfna_old                   -> display3
 </PRE>
*/
public GoesCollaboration(String args[])
       throws VisADException, RemoteException {

  if (args.length > 0) {
    // this is a client

    // try to connect to RemoteServer
    String domain = "//" + args[0] + "/GoesCollaboration";
    try {
      client_server = (RemoteServer) Naming.lookup(domain);
    }
    catch (MalformedURLException e) {
      System.out.println("Cannot connect to server");
      System.exit(0);
    }
```

```
    catch (NotBoundException e) {
      System.out.println("Cannot connect to server");
      System.exit(0);
    }
    catch (AccessException e) {
      System.out.println("Cannot connect to server");
      System.exit(0);
    }
    catch (RemoteException e) {
      System.out.println("Cannot connect to server");
      System.exit(0);
    }
  }
  else { // args.length == 0
    // this is a server

    /* CTR: 30 Sep 1998 */
    // check for the existence of necessary data files
    {
      File f1 = new File("data_obs_1.dat");
      File f2 = new File("goesrtcf");
      if (!f1.exists() || !f2.exists()) {
        System.out.println("This program requires the data files " +
                           "\"data_obs_1.dat\"");
        System.out.println("and \"goesrtcf\", available at:");
System.out.println("   ftp://demedici.ssec.wisc.edu/pub/visad-2.0/" +
                           "paoloa-files.tar.Z");
        System.exit(1);
      }
      if (!f2.exists()) {
        System.out.println("");
        System.exit(2);
      }
    }

    // try to set up a RemoteServer
    server_server = new RemoteServerImpl();
    try {
      Naming.rebind("//:/GoesCollaboration", server_server);
    }
    catch (MalformedURLException e) {
System.out.println("Cannot set up server - running as stand-alone");
      server_server = null;
    }
    catch (AccessException e) {
System.out.println("Cannot set up server - running as stand-alone");
      server_server = null;
    }
    catch (RemoteException e) {
System.out.println("Cannot set up server - running as stand-alone");
      server_server = null;
    }
  }
}

/** set up as server */
void setupServer() throws VisADException, RemoteException {

  //
  // construct function domain sampling Sets
```

```
//

// construct 1-D Sets
Set linear18 = new Linear1DSet(1.0, 18.0, 18);
Set linear19 = new Linear1DSet(1.0, 19.0, 19);
Set linear40 = new Linear1DSet(1.0, 40.0, 40);

// construct 2-D Set
Set linear40x18 = new Linear2DSet(1.0, 40.0, 40, 1.0, 18.0, 18);

//
// construct MathTypes for Data objects
//

// construct RealTypes used as Function domains
// with null Units but non-null default Sets (for
// function domain samplings)
nchan = new RealType("nchan", null, linear18);
indx = new RealType("indx", null, linear19);
nl = new RealType("nl", null, linear40);

// construct RealTypes used as Function ranges
// or for simple Real values, with null Units
// and null default Sets
tbc = new RealType("tbc", null, null);
tbc_d = new RealType("tbc_d", null, null);
wfn = new RealType("wfn", null, null);
pres = new RealType("pres", null, null);
temp = new RealType("temp", null, null);
mixr = new RealType("mixr", null, null);
ozone = new RealType("ozone", null, null);
pressure = new RealType("pressure", null, null);
data_real = new RealType("data_real", null, null);
diff = new RealType("diff", null, null);

// construct RealTupleType used as a Function domain
// with non-null default Set
RealTupleType nl_nchan = new RealTupleType(nl, nchan, null,
                                           linear40x18);

// construct FunctionTypes
FunctionType obs_data = new FunctionType(indx, data_real);
FunctionType wfn_big = new FunctionType(nl_nchan,
                                        new RealTupleType(wfn, tbc));
FunctionType tbc_array_dif = new FunctionType(nchan, tbc_d);
FunctionType wfn_array = new FunctionType(nl_nchan, wfn);
FunctionType temp_array = new FunctionType(nl, temp);
FunctionType mixr_array = new FunctionType(nl, mixr);
FunctionType ozone_array = new FunctionType(nl, ozone);
FunctionType pres_array = new FunctionType(nl, pressure);

//
// construct Data objects and DataReferences to them
//

// construct weighting function Data object and DataReference
FlatField wfna = new FlatField(wfn_array);
wfna_ref = new DataReferenceImpl("wfna");
wfna_ref.setData(wfna);
```

```
// construct temperature profile Data object and DataReference
FlatField tempa = new FlatField(temp_array);
tempa_ref = new DataReferenceImpl("tempa");
tempa_ref.setData(tempa);

// construct mixing ratio profile Data object and DataReference
FlatField mixra = new FlatField(mixr_array);
mixra_ref = new DataReferenceImpl("mixra");
mixra_ref.setData(mixra);

// construct ozone profile Data object and DataReference
FlatField ozonea = new FlatField(ozone_array);
ozonea_ref = new DataReferenceImpl("ozonea");
ozonea_ref.setData(ozonea);

// construct pressure profile Data object and DataReference
FlatField presa = new FlatField(pres_array);
presa_ref = new DataReferenceImpl("presa");
presa_ref.setData(presa);

// construct weighting function difference Data object
// and DataReference
FlatField diff_col = new FlatField(wfn_array);
diff_col_ref = new DataReferenceImpl("diff_col");
diff_col_ref.setData(diff_col);

// construct brightness temperature error Data object
// and DataReference
FlatField diff_DATA = new FlatField(tbc_array_dif);
diff_ref = new DataReferenceImpl("diff");
diff_ref.setData(diff_DATA);

// construct zero line Data object and DataReference
FlatField zero_line = new FlatField(tbc_array_dif);
zero_line_ref = new DataReferenceImpl("zero_line");
zero_line_ref.setData(zero_line);

// construct brightness temperature error root mean square
// Data object and DataReference
Real smr = new Real(tbc_d);
smr_ref = new DataReferenceImpl("smr");
smr_ref.setData(smr);

// construct observed brightness temperature Data object
// and DataReference
FlatField real_tbc = new FlatField(obs_data);
real_tbc_ref = new DataReferenceImpl("real_tbc");
real_tbc_ref.setData(real_tbc);

// construct compound weighting function Data object
// and DataReference
FlatField wfnb = new FlatField(wfn_big);
wfnb_ref = new DataReferenceImpl("wfnb");
wfnb_ref.setData(wfnb);

// construct saved weighting function Data object
// and DataReference
FlatField wfna_old = new FlatField(wfn_array);
wfna_old_ref = new DataReferenceImpl("wfna_old");
wfna_old_ref.setData(wfna);
```

```
//
// construct DataReference objects linked to VisADSliders (the
// JSlider constructors will construct Real data objects for
// these, so there is no point in constructing Real data objects
// here)
//

// DataReference for zenith angle
gzen_ref = new DataReferenceImpl("gzen");

// DataReference for skin temperature
tskin_ref = new DataReferenceImpl("tskin");

// DataReference for index into model atmospheres
in_dx_ref = new DataReferenceImpl("in_dx");

// DataReference used to trigger copying wfna to wfna_old
DataReference save_config_ref = new DataReferenceImpl("save_config");


// set up Displays for server
DisplayImpl[] displays = new DisplayImpl[4];
setupDisplays(displays);
if (server_server != null) {
  for (int i = 0; i < displays.length; i++) {
    server_server.addDisplay(new RemoteDisplayImpl(displays[i]));
  }
}

// set up user interface
setupUI(displays, in_dx_ref, save_config_ref, gzen_ref, tskin_ref);


// initialize zero reference line for brightness temperature errors
double[][] zero_line_x = zero_line.getValues();
for (int i=0; i<zero_line_x[0].length; i++) zero_line_x[0][i] = 0.0;
zero_line.setSamples(zero_line_x);


// make sure Data are initialized
new Delay(1000);
gzen_ref.incTick();
save_config_ref.incTick();
new Delay(1000);


//
// construct computational Cells and links to DataReferences
// that trigger them
//

// construct a real_tbcCell
real_tbcCell real_tbc_cell = new real_tbcCell();
real_tbc_cell.addReference(in_dx_ref);
new Delay(500);

// construct a wfnbCell
wfnbCell wfnb_cell = new wfnbCell();
```

```
  wfnb_cell.addReference(gzen_ref);
  wfnb_cell.addReference(tskin_ref);
  wfnb_cell.addReference(tempa_ref);
  wfnb_cell.addReference(mixra_ref);
  wfnb_cell.addReference(ozonea_ref);
  wfnb_cell.addReference(presa_ref);
  new Delay(500);

  // construct a wfnaCell
  wfnaCell wfna_cell = new wfnaCell();
  wfna_cell.addReference(wfnb_ref);
  wfna_cell.addReference(real_tbc_ref);
  new Delay(500);

  // construct a wfna_oldCell
  wfna_oldCell wfna_old_cell = new wfna_oldCell();
  wfna_old_cell.addReference(save_config_ref);
  new Delay(500);

  // construct a diff_colCell
  diff_colCell diff_col_cell = new diff_colCell();
  diff_col_cell.addReference(wfna_ref);
  diff_col_cell.addReference(wfna_old_ref);
  new Delay(500);


  if (server_server != null) {
    // set RemoteDataReferenceImpls in RemoteServer
    RemoteDataReferenceImpl[] refs =
      new RemoteDataReferenceImpl[4];
    refs[0] =
      new RemoteDataReferenceImpl((DataReferenceImpl) gzen_ref);
    refs[1] =
      new RemoteDataReferenceImpl((DataReferenceImpl) tskin_ref);
    refs[2] =
      new RemoteDataReferenceImpl((DataReferenceImpl) in_dx_ref);
    refs[3] =
      new RemoteDataReferenceImpl((DataReferenceImpl) save_config_ref);

    server_server.setDataReferences(refs);
  }


  // make sure Data are initialized (again)
  new Delay(1000);
  gzen_ref.incTick();
  save_config_ref.incTick();

}

/** set up as client */
void setupClient() throws VisADException, RemoteException {

  //
  // get RemoteDataReferences
  //

  RemoteDataReference[] refs = client_server.getDataReferences();
  if (refs == null) {
    System.out.println("Cannot connect to server");
```

```
    System.exit(0);
  }

  gzen_ref = refs[0];
  tskin_ref = refs[1];
  in_dx_ref = refs[2];
  DataReference save_config_ref = refs[3];

  // set up Displays for client
  DisplayImpl[] displays = new DisplayImpl[4];
  displays[0] =
    new DisplayImplJ3D(client_server.getDisplay("display1"));
  displays[1] =
    new DisplayImplJ3D(client_server.getDisplay("display2"));
  displays[2] =
    new DisplayImplJ3D(client_server.getDisplay("display3"));
  displays[3] =
    new DisplayImplJ3D(client_server.getDisplay("display4"));

  // set up user interface
  setupUI(displays, in_dx_ref, save_config_ref, gzen_ref, tskin_ref);

}


/** set up Displays; return constructed Displays in displays array */
void setupDisplays(DisplayImpl[] displays)
     throws VisADException, RemoteException {

  //
  // construct Displays and link to Data objects
  //

  // construct Display 1 (using default DisplayRenderer);
  // the text name is used only for debugging
  DisplayImplJ3D display1 = new DisplayImplJ3D("display1");
  // construct ScalarMaps for Display 1;
  // explicitly set data range for nl values (in order to
  // invert scale)
  ScalarMap map1nl = new ScalarMap(nl, Display.YAxis);
  map1nl.setRange(40.0, 1.0);
  display1.addMap(map1nl);
  // setRange is not invoked for other ScalarMaps - they will
  // use auto-scaling from actual data values
  display1.addMap(new ScalarMap(nchan, Display.XAxis));
  display1.addMap(new ScalarMap(wfn, Display.Green));
  display1.addMap(new ScalarMap(wfn, Display.ZAxis));
  display1.addMap(new ConstantMap(0.5f, Display.Red));
  display1.addMap(new ConstantMap(0.5f, Display.Blue));

  GraphicsModeControl mode1 = display1.getGraphicsModeControl();
  mode1.setScaleEnable(true);

  // link weighting function Data object to display1
  // (using default DataRenderer and a null array of ConstantMaps)
  display1.addReference(wfna_ref);


  // construct Display 2 and its ScalarMaps (using non-default
  // 2-D DisplayRenderer)
```

```
DisplayImplJ3D display2 =
  new DisplayImplJ3D("display2", new TwoDDisplayRendererJ3D());
// explicitly set data range for nl values (in order to
// invert scale)
ScalarMap map2nl = new ScalarMap(nl, Display.YAxis);
map2nl.setRange(40.0, 1.0);
display2.addMap(map2nl);
// map temp, mixr and ozone to XAxis and
// set axis scale colors
ScalarMap map2temp = new ScalarMap(temp, Display.XAxis);
display2.addMap(map2temp);
map2temp.setScaleColor(new float[] {1.0f, 0.0f, 0.0f});
ScalarMap map2mixr = new ScalarMap(mixr, Display.XAxis);
display2.addMap(map2mixr);
map2mixr.setScaleColor(new float[] {0.0f, 1.0f, 0.0f});
ScalarMap map2ozone = new ScalarMap(ozone, Display.XAxis);
display2.addMap(map2ozone);
map2ozone.setScaleColor(new float[] {0.0f, 0.0f, 1.0f});
display2.addMap(new ScalarMap(pressure, Display.XAxis));

GraphicsModeControl mode2 = display2.getGraphicsModeControl();
mode2.setLineWidth(2.0f);
mode2.setScaleEnable(true);

// color temperature profile red
ConstantMap[] tmaps = {new ConstantMap(1.0f, Display.Red),
                       new ConstantMap(0.0f, Display.Green),
                       new ConstantMap(0.0f, Display.Blue)};

// color mixing ratio profile green
ConstantMap[] mmaps = {new ConstantMap(0.0f, Display.Red),
                       new ConstantMap(1.0f, Display.Green),
                       new ConstantMap(0.0f, Display.Blue)};

// color ozone profile blue
ConstantMap[] omaps = {new ConstantMap(0.0f, Display.Red),
                       new ConstantMap(0.0f, Display.Green),
                       new ConstantMap(1.0f, Display.Blue)};

// color pressure profile white
ConstantMap[] pmaps = {new ConstantMap(1.0f, Display.Red),
                       new ConstantMap(1.0f, Display.Green),
                       new ConstantMap(1.0f, Display.Blue)};

// enable direct manipulation for temperature, mixing ratio
// and ozone profiles; do not enable direct manipulation for
// pressure;
// note that addReferences rather than addReference is
// invoked for non-default DataRenderers (in this case,
// DirectManipulationRendererJ3D);
// note also that addReference and addReferences may take
// an array of ConstantMaps that apply only to one Data
// object
display2.addReferences(new DirectManipulationRendererJ3D(),
      tempa_ref, tmaps);
display2.addReferences(new DirectManipulationRendererJ3D(),
      mixra_ref, mmaps);
display2.addReferences(new DirectManipulationRendererJ3D(),
      ozonea_ref, omaps);
display2.addReference(presa_ref, pmaps);
```

```
  // construct Display 3 and its ScalarMaps
  DisplayImplJ3D display3 = new DisplayImplJ3D("display3");
  // explicitly set data range for nl values (in order to
  // invert scale)
  ScalarMap map3nl = new ScalarMap(nl, Display.YAxis);
  map3nl.setRange(40.0, 1.0);
  display3.addMap(map3nl);
  display3.addMap(new ScalarMap(nchan, Display.XAxis));
  display3.addMap(new ScalarMap(wfn, Display.ZAxis));
  display3.addMap(new ScalarMap(wfn, Display.Green));
  display3.addMap(new ConstantMap(0.5f, Display.Red));
  display3.addMap(new ConstantMap(0.5f, Display.Blue));

  GraphicsModeControl mode3 = display3.getGraphicsModeControl();
  mode3.setScaleEnable(true);

  // link weighting function difference Data object to display3
  display3.addReference(diff_col_ref);


  // construct Display 4 and its ScalarMaps (using non-default
  // 2-D DisplayRenderer)
  DisplayImplJ3D display4 =
    new DisplayImplJ3D("display4", new TwoDDisplayRendererJ3D());
  display4.addMap(new ScalarMap(nchan, Display.XAxis));
  // explicitly set data range for tbc_d values
  ScalarMap map4tbc_d = new ScalarMap(tbc_d, Display.YAxis);
  map4tbc_d.setRange(-40.0, 40.0);
  display4.addMap(map4tbc_d);

  // set pointSize = 5 in display4 to make single Real value smr
  //   easily visible
  GraphicsModeControl mode4 = display4.getGraphicsModeControl();
  mode4.setPointSize(5.0f);
  mode4.setLineWidth(2.0f);
  mode4.setScaleEnable(true);

  // link brightness temperature error, zero line and brightness
  // temperature error root mean square Data objects to display4
  display4.addReference(diff_ref);
  display4.addReference(zero_line_ref);
  display4.addReference(smr_ref);

  // return DisplayImpls
  displays[0] = display1;
  displays[1] = display2;
  displays[2] = display3;
  displays[3] = display4;
}


/** construct user interface using JFC */
void setupUI(DisplayImpl[] displays, DataReference in_dx_ref,
             DataReference save_config_ref, DataReference gzen_ref,
             DataReference tskin_ref)
     throws VisADException, RemoteException {

  //
```

```
// construct JFC user interface with JSliders linked to
// Data objects, and embed Displays into JFC JFrame
//

// create a JFrame
JFrame frame = new JFrame("GoesCollaboration");
WindowListener l = new WindowAdapter() {
  public void windowClosing(WindowEvent e) {System.exit(0);}
};
frame.addWindowListener(l);
frame.setSize(WIDTH, HEIGHT);
frame.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
frame.setLocation(screenSize.width/2 - WIDTH/2,
                  screenSize.height/2 - HEIGHT/2);

// create big_panel JPanel in frame
JPanel big_panel = new JPanel();
big_panel.setLayout(new BoxLayout(big_panel, BoxLayout.X_AXIS));
big_panel.setAlignmentY(JPanel.TOP_ALIGNMENT);
big_panel.setAlignmentX(JPanel.LEFT_ALIGNMENT);
frame.getContentPane().add(big_panel);

// create left hand side JPanel for sliders and text
JPanel left = new JPanel(); // FlowLayout and double buffer
left.setLayout(new BoxLayout(left, BoxLayout.Y_AXIS));
left.setAlignmentY(JPanel.TOP_ALIGNMENT);
left.setAlignmentX(JPanel.LEFT_ALIGNMENT);
big_panel.add(left);

// construct JLabels
// (JTextArea does not align in BoxLayout well, so use JLabels)
left.add(new JLabel("Interactive GOES satellite sounding " +
                    "retrieval"));
left.add(new JLabel("using VisAD  -  see:"));
left.add(new JLabel("  "));
left.add(new JLabel("  http://www.ssec.wisc.edu/~billh/visad.html"));
left.add(new JLabel("  "));
left.add(new JLabel("for more information about VisAD."));
left.add(new JLabel("  "));
left.add(new JLabel("Bill Hibbard, Paolo Antonelli and Bob Aune"));
left.add(new JLabel("Space Science and Engineering Center"));
left.add(new JLabel("University of Wisconsin - Madison"));
left.add(new JLabel("  "));
left.add(new JLabel("  "));
left.add(new JLabel("Move index slider to retrieve a new model"));
left.add(new JLabel("atmosphere."));
left.add(new JLabel("  "));
left.add(new JLabel("Touch ref. conf. slider to save a new"));
left.add(new JLabel("reference for weighting function " +
                    "difference."));
left.add(new JLabel("  "));
left.add(new JLabel("Move zenith angle and skin T sliders to"));
left.add(new JLabel("to modify atmosphere conditions."));
left.add(new JLabel("  "));
left.add(new JLabel("Rotate scenes with left mouse button."));
left.add(new JLabel("  "));
left.add(new JLabel("Redraw temperature, water vapor and ozone " +
                    "with"));
left.add(new JLabel("right mouse button to modify model " +
```

```
                    "atmosphere."));
left.add(new JLabel("  "));
left.add(new JLabel("  "));

// create sliders JPanel
JPanel sliders = new JPanel();
sliders.setName("GoesCollaboration Sliders");
sliders.setFont(new Font("Dialog", Font.PLAIN, 12));
sliders.setLayout(new BoxLayout(sliders, BoxLayout.Y_AXIS));
sliders.setAlignmentY(JPanel.TOP_ALIGNMENT);
sliders.setAlignmentX(JPanel.LEFT_ALIGNMENT);
left.add(sliders);

// construct VisADSliders linked to Real Data objects and embedded
// in sliders JPanel
sliders.add(new VisADSlider("index", 1, 2234, 1, 1.0, in_dx_ref,
                            RealType.Generic));
sliders.add(new JLabel("  "));
sliders.add(new VisADSlider("save as ref. conf.?", 0, 1000, 0, 1.0,
                            save_config_ref,  RealType.Generic));
sliders.add(new JLabel("  "));
sliders.add(new VisADSlider("zenith angle (deg)", 0, 65, 35, 1.0,
                            gzen_ref, RealType.Generic));
sliders.add(new JLabel("  "));
sliders.add(new VisADSlider("skin T (K)", 250, 340, 300, 1.0,
                            tskin_ref, RealType.Generic));

// construct JPanel and sub-panels for Displays
JPanel display_panel = new JPanel();
display_panel.setLayout(new BoxLayout(display_panel,
                                      BoxLayout.X_AXIS));
display_panel.setAlignmentY(JPanel.TOP_ALIGNMENT);
display_panel.setAlignmentX(JPanel.LEFT_ALIGNMENT);
big_panel.add(display_panel);

JPanel display_left = new JPanel();
display_left.setLayout(new BoxLayout(display_left,
                                     BoxLayout.Y_AXIS));
display_left.setAlignmentY(JPanel.TOP_ALIGNMENT);
display_left.setAlignmentX(JPanel.LEFT_ALIGNMENT);
display_panel.add(display_left);

JPanel display_right = new JPanel();
display_right.setLayout(new BoxLayout(display_right,
                                      BoxLayout.Y_AXIS));
display_right.setAlignmentY(JPanel.TOP_ALIGNMENT);
display_right.setAlignmentX(JPanel.LEFT_ALIGNMENT);
display_panel.add(display_right);

// get Display panels
JPanel panel1 = (JPanel) displays[0].getComponent();
JPanel panel2 = (JPanel) displays[1].getComponent();
JPanel panel3 = (JPanel) displays[2].getComponent();
JPanel panel4 = (JPanel) displays[3].getComponent();

// make borders for Displays and embed in display_panel JPanel
Border etchedBorder10 =
  new CompoundBorder(new EtchedBorder(),
                     new EmptyBorder(10, 10, 10, 10));
panel1.setBorder(etchedBorder10);
```

```
  panel2.setBorder(etchedBorder10);
  panel3.setBorder(etchedBorder10);
  panel4.setBorder(etchedBorder10);

  // make labels for Displays
  JLabel display1_label = new JLabel("weighting function");
  JLabel display1a_label =
    new JLabel("vertical level (Y) vs channel (X)");
  JLabel display2_label = new JLabel("model atmosphere profile");
  JLabel display2a_label =
    new JLabel("temperature (red), ozone (blue),");
  JLabel display2b_label =
    new JLabel("water vapor (green), pressure (white)");
  JLabel display3_label = new JLabel("weighting function difference");
  JLabel display3a_label =
     new JLabel("vertical level (Y) vs channel (X)");
  JLabel display4_label = new JLabel("brightness temperature errors");
  JLabel display4a_label = new JLabel("with zero reference line and");
  JLabel display4b_label =
     new JLabel("root mean square error (single point)");

  // embed Displays and their labels in display_panel JPanel
  display_left.add(panel1);
  display_left.add(display1_label);
  display_left.add(display1a_label);
  display_left.add(panel2);
  display_left.add(display2_label);
  display_left.add(display2a_label);
  display_left.add(display2b_label);
  display_right.add(panel3);
  display_right.add(display3_label);
  display_right.add(display3a_label);
  display_right.add(panel4);
  display_right.add(display4_label);
  display_right.add(display4a_label);
  display_right.add(display4b_label);

  // make the JFrame visible
  frame.setVisible(true);
}


/** get observed brightness temperatures, as well as temperature,
    water-vapor mixing-ratio, ozone and pressure profiles */
class real_tbcCell extends CellImpl {

  public void doAction() throws VisADException, RemoteException {
    // get index into model atmospheres
    int in_dx = (int) ((Real) in_dx_ref.getData()).getValue();
    if (in_dx < 1 || in_dx > 2234) return;

    // read observed brightness temperatures from data_obs_1.dat
    float[][] data_b = new float[1][19];
    re_read_1_c(in_dx, data_b[0]);
    ((FlatField) real_tbc_ref.getData()).setSamples(data_b);

    // obtain climatological temperature, water-vapor mixing-ratio,
    // and ozone mixing-ratio profiles by interpolating in month
    // and latitude amongst the FASCODE model atmospheres;
    // also get fixed pressure levels
```

```
    float lat = data_b[0][18];
    int month = 6;
    float[][] t_x = new float[1][40];
    float[][] m_x = new float[1][40];
    float[][] o_x = new float[1][40];
    float[][] p_x = new float[1][40];
    get_profil_c(lat, month, t_x[0], m_x[0], o_x[0], p_x[0]);

    ((FlatField) tempa_ref.getData()).setSamples(t_x);
    ((FlatField) mixra_ref.getData()).setSamples(m_x);
    ((FlatField) ozonea_ref.getData()).setSamples(o_x);
    ((FlatField) presa_ref.getData()).setSamples(p_x);
  }
}

/** compute weighting function of channel versus vertical level */
class wfnbCell extends CellImpl {

  public void doAction() throws VisADException, RemoteException {
    // get zenith angle and skin temperature
    float gzen = (float) ((Real) gzen_ref.getData()).getValue();
    float tskin = (float) ((Real) tskin_ref.getData()).getValue();

    // compute weighting function of channel versus vertical level
    float[][] t_x = Set.doubleToFloat(((FlatField)
                        tempa_ref.getData()).getValues());
    float[][] m_x = Set.doubleToFloat(((FlatField)
                        mixra_ref.getData()).getValues());
    float[][] o_x = Set.doubleToFloat(((FlatField)
                        ozonea_ref.getData()).getValues());
    float[][] p_x = Set.doubleToFloat(((FlatField)
                        presa_ref.getData()).getValues());
    float[][] wfn = new float[2][40*18];
    goesrte_2_c(gzen, tskin, t_x[0], m_x[0], o_x[0], p_x[0],
                wfn[0], wfn[1]);
    ((FlatField) wfnb_ref.getData()).setSamples(wfn);
  }
}

/** compute brightness temperature errors and root mean square */
class wfnaCell extends CellImpl {

  public void doAction() throws VisADException, RemoteException {
    // compute brightness temperature errors
    float[][] t_x = new float[1][];
    float[][] wfn =
      Set.doubleToFloat(((FlatField) wfnb_ref.getData()).getValues());
    t_x[0] = wfn[0];
    ((FlatField) wfna_ref.getData()).setSamples(t_x);
    float[][] real_tbc_x = Set.doubleToFloat(((FlatField)
                           real_tbc_ref.getData()).getValues());
    float[][] diff_DATA_x = new float[1][18];
    float squ_mod = 0.0f;
    for (int c=0; c<18; c++) {
      diff_DATA_x[0][c] = wfn[1][0 + 40 * c] - real_tbc_x[0][c];
      squ_mod += diff_DATA_x[0][c] * diff_DATA_x[0][c] / 18.0f;
    }
    ((FlatField) diff_ref.getData()).setSamples(diff_DATA_x);

    // smr is root mean square of brightness temperature errors
```

```
      smr_ref.setData(new Real(tbc_d, Math.sqrt(squ_mod)));
    }
  }

  /** save a copy of wfna in wfna_old */
  class wfna_oldCell extends CellImpl {

    public void doAction() throws VisADException, RemoteException {
      // save a copy of wfna in wfna_old (i.e., wfna_old = wfna)
      wfna_old_ref.setData(
        (FlatField) ((FlatField) wfna_ref.getData()).clone());
    }
  }

  /** compute diff_col = wfna - wfna_old */
  class diff_colCell extends CellImpl {

    public void doAction() throws VisADException, RemoteException {
      // compute diff_col = wfna - wfna_old
      diff_col_ref.setData(
        wfna_ref.getData().subtract(wfna_old_ref.getData()));
    }
  }

  /** native method declarations, to Fortran via C */
  private native void re_read_1_c(int i, float[] data_b);

  private native void goesrte_2_c(float gzen, float tskin, float[] t,
                                  float[] w, float[] c, float[] p,
                                  float[] wfn, float[] tbcx);

  private native void get_profil_c(float rlat, int imon, float[] tpro,
                                   float[] wpro, float[] opro,
                                   float[] pref);

}
```