SC/2.3

Starlink Project
Starlink Cookbook 2.3

A.C. Davenhall

1st October 1997

The DX Cookbook

Abstract

IBM DX (Data Explorer) is the data-visualisation package recommended by Starlink, particularly for the visualisation of three-dimensional scalar and vector data. This cookbook provides a set of simple recipes for performing common operations using DX. No prior knowledge of DX is required. You can use the recipes as they are provided, if they are suitable for your purposes, or alternatively you can use them as a starting point for generating your own visualisations.

The cookbook also contains a brief summary of the DX data model. It is useful to have at least some understanding of this data model in order to use DX effectively.

This edition of the cookbook applies to Version 3.1 of DX.

Contents

List of Figures

Revision history

(1) 26th January 1996: Version 1 (ACD).

(2) 28th February 1997: Version 2. Minor changes (ACD).

(3) 1st October 1997: Version 3. Added Part II 'Extended Recipes'. Also made a few other minor changes (ACD).

1 Introduction

IBM DX (Data Explorer) is the data visualisation package recommended by Starlink, particularly for the visualisation of three-dimensional scalar and vector data. It is a powerful and flexible package capable of generating sophisticated visualisations of complex data. However, a necessary consequence of this complexity is that it is non-trivial to use and it is necessary to invest a certain amount of time to learn to use it effectively. This document is an aid to learning to use DX. It is not a user's guide or a tutorial, but rather provides a set of 'recipes' for performing simple visualisations of the sort which you may require. You can either use the recipes 'as is' if they do exactly what you want, or (more likely) vary them slightly to meet your requirements. This edition of the cookbook applies to Version 3.1 of DX.

Like any technical subject, scientific visualisation has its own collection of specialised jargon and techniques. SG/8 *An Introduction to Visualisation Software for Astronomy*[4] gives an overview and introduction to scientific visualisation relevant to astronomy. If you are already familiar with visualisation techniques you probably do not need to refer to it. However, if you are new to visualisation and are not sure which techniques may be suitable for your data, you might find it useful.

DX is essentially a tool which allows you to write programs or scripts which generate some particular visualisation of a dataset. Though it is possible to use pre-existing DX programs, most of the time you will use DX to create, modify and use your own programs. DX programs can be written using a text-based scripting language, which is not dissimilar to conventional programming and scripting languages. However, this scripting language is not the usual way to write DX programs, and it is not discussed in this document. Rather, DX programs are usually written using a 'visual programming editor'. Icons representing modules to perform some function (for example, reading a file, smoothing an image, plotting an image etc.) are positioned on a canvas[1] and joined by lines representing the flow of data between modules. The assemblage so generated performs the required visualisation (typically it will start by reading a data file and end by generating an image). These assemblages are known as 'networks' or 'visual programs'. Part I of the cookbook contains several examples of these networks; see, for example, Figure 3. Though DX networks superficially resemble flow charts, it is important to realise that they are quite different. The lines in a DX network show the flow of *data* through the system, *not* the flow of *control* as the network executes[2]. Obviously, once you have created a network with the visual programing editor it can be saved to disk as a file and subsequently reloaded; it is not necessary to create a network *ab initio* each time you use DX. The recipes in this cookbook concentrate exclusively on using the visual programming editor to generate networks.

The structure of the cookbook is:

Part I – simple recipes for simple, common visualisations,

Part II – extended recipes with some more complex visualisations,

Part III – introduction to the DX data model.

[1]A blank area of a window used for constructing diagrams.

[2]If you are familiar with the Yourdon-de Marco Structured Systems Analysis technique, or other similar methods, you will recognise that DX networks are very similar to the data flow diagrams used in these techniques.

In order to use DX effectively it is useful to have some understanding of the data model which it uses. You should really read all of Part III (it is only a few pages), but at the very least you should read Section 16.

Starlink have provided a set of enhancements to DX, called SX. These enhancements fill a few minor omissions in the functionality of the basic DX and also package some common functions to make them easier to use. If DX is available at a Starlink site then usually the Starlink enhancements will also be available automatically. Where appropriate this cookbook refers to the enhancements as well as basic DX.

The use of DX on Starlink systems, and the Starlink enhancements, are documented in SUN/203 *DX — IBM Data Explorer for Data Visualisation*[11]. There are also various manuals provided with DX (see Table 1 of SUN/203 for a list). In particular the *QuickStart Guide*[1] gives a further set of tutorials for generating common visualisations.

2 Starting DX

No special privileges or quotas are required to run DX. DX (and, indeed, visualisation software in general) tends to be quite profligate in its use of computing resources. Files containing generated images can require a significant amount of disk space, and in particular files containing animations can be extremely large. Generating visualisations is computationally intensive and can require substantial amounts of computer memory. Therefore it is sensible to run DX on the computer with the fastest processor and largest physical memory to which you have convenient access.

DX is available for Digital Unix/alpha and Solaris/Sun. Your site manager should be able to advise which is available at your site.

To use DX you need a display capable of receiving X-output (typically an X-terminal or a workstation console). DX will run on a black and white display, but realistically you need a colour display[3].

To start DX, ensure that your display is configured to receive X-output and then simply type:

```
% dx
```

The following message should appear on your command terminal:

```
Starting DX user interface
```

and a new window (technically the canvas for the DX visual programming editor) should appear. If DX fails to start properly then consult your site manager who should be able to advise. The most likely reason is that DX is not installed at your site. The Starlink extensions to DX, SX, should be automatically available when you start DX. If they are not available then check that your site manager has installed them.

[3] For example, with a black and white display the various icons for modules in the visual programming editor are only partly visible.

2.1 Starting DX without the Starlink enhancements

It is possible to start DX without the Starlink enhancements, SX. Again ensure that your terminal is configured to receive X-output. Then type:

```
% unalias   dx
```

Followed by:

```
% dx
```

DX should now start, as above, but the Starlink enhancements will not be available.

2.2 Re-instating the Starlink enhancements

If you have disabled the Starlink enhancements, SX, then you can re-instate them by typing:

```
% alias dx tcsh -c '"source $SX_DIR/dx.csh"'
```

Then start DX as normal: ensure that your terminal is configured to receive X-output and type:

```
% dx
```

DX should now start with the Starlink enhancements available.

Part I
Simple Recipes

3 Introduction

This section presents a set of recipes for performing common operations using DX.

- The first recipe (Section 4) introduces the basic operation of the DX visual programming editor. You should follow this example to learn the minimum basics of using DX.
- The second and third examples (Sections 5 and 6 respectively) describe how to prepare gridded and particle data for input to DX. Before an external data file can be imported into DX you must prepare a header file describing it. These recipes describe this process.
- The remaining recipes present simple networks for performing common types of visualisations. Though simple, the networks are complete and self-contained; they include all the steps that are necessary to perform the visualisation. In these cases, in a sense, the network is the recipe.

3.1 Copies of the recipe networks

Copies of the recipe networks, data files, example programs and other files referred to in this section can be found on Starlink systems in directory:

```
/star/examples/sc2
```

and on non-Starlink systems in directory:

```
$STARLINK_DIR/examples/sc2
```

The most convenient way to access these files is probably to copy them into your current directory before attempting to read them or load them into DX. The examples in subsequent sections assume that copies of the files are available in your current directory.

3.2 Further examples

The IBM Manual *QuickStart Guide*[1] provided with DX contains numerous further example networks and an extensive discussion of importing data files into DX. All the example networks described in the *QuickStart Guide* are included with the release of DX.

4 Basic Use of DX

This recipe introduces the basic use of DX. It covers creating and running a simple DX network. Only the very minimum set of features needed to create and run a network are mentioned; there are many features and facilities which are omitted. See the *QuickStart Guide*[1] and other IBM DX manuals for further details (the IBM manuals are listed in Table 1 of SUN/203[11]). The best way to use this recipe is probably to have a terminal on which you can run DX to hand and to work through the recipe trying the various points.

Unless otherwise noted, throughout this cookbook 'click' means click on the leftmost mouse button.

(1) To start DX (see also Section 2, above), ensure that your display is configured to receive X-output and then simply type:

```
% dx
```

The following message should appear on your command terminal:

```
Starting DX user interface
```

and a new window similar to Figure (1) should appear.

(2) Figure (1) shows the main window for the Visual Program Editor (VPE). You will use it to construct DX networks which in turn will generate visualisations.

- The large blank area is the canvas where the network will be constructed.
- A DX network consists of modules, represented by icons, each performing some function. The collection of modules is divided into categories of similar modules. On the left hand side of the VPE are two boxes containing lists of items. The top box, labelled 'Categories:', lists all the different categories of modules. The current category is highlighted (it is 'Annotation' in Figure (1)). The lower box, labelled 'Annotation Tools', shows all the modules in the current category.

  To choose another category simply click on the category required. The new category will be highlighted and both the label and the contents of the lower box will be changed to reflect the new choice.

(3) To place a module on the canvas (for subsequent inclusion in a network) proceed as follows:

(a) select the appropriate category, so that the required module appears in the lower box,

(b) click on the required module in the lower box, and it will be highlighted,

(c) move the cursor to the desired position in the canvas and click again. The module will appear as a rectangular icon (see, for example, Figure 3).

(4) If you want to move a module to another position in the canvas (for example, because you have inadvertently put it in the wrong place):

(a) click in the body of the module and hold down the leftmost mouse button,

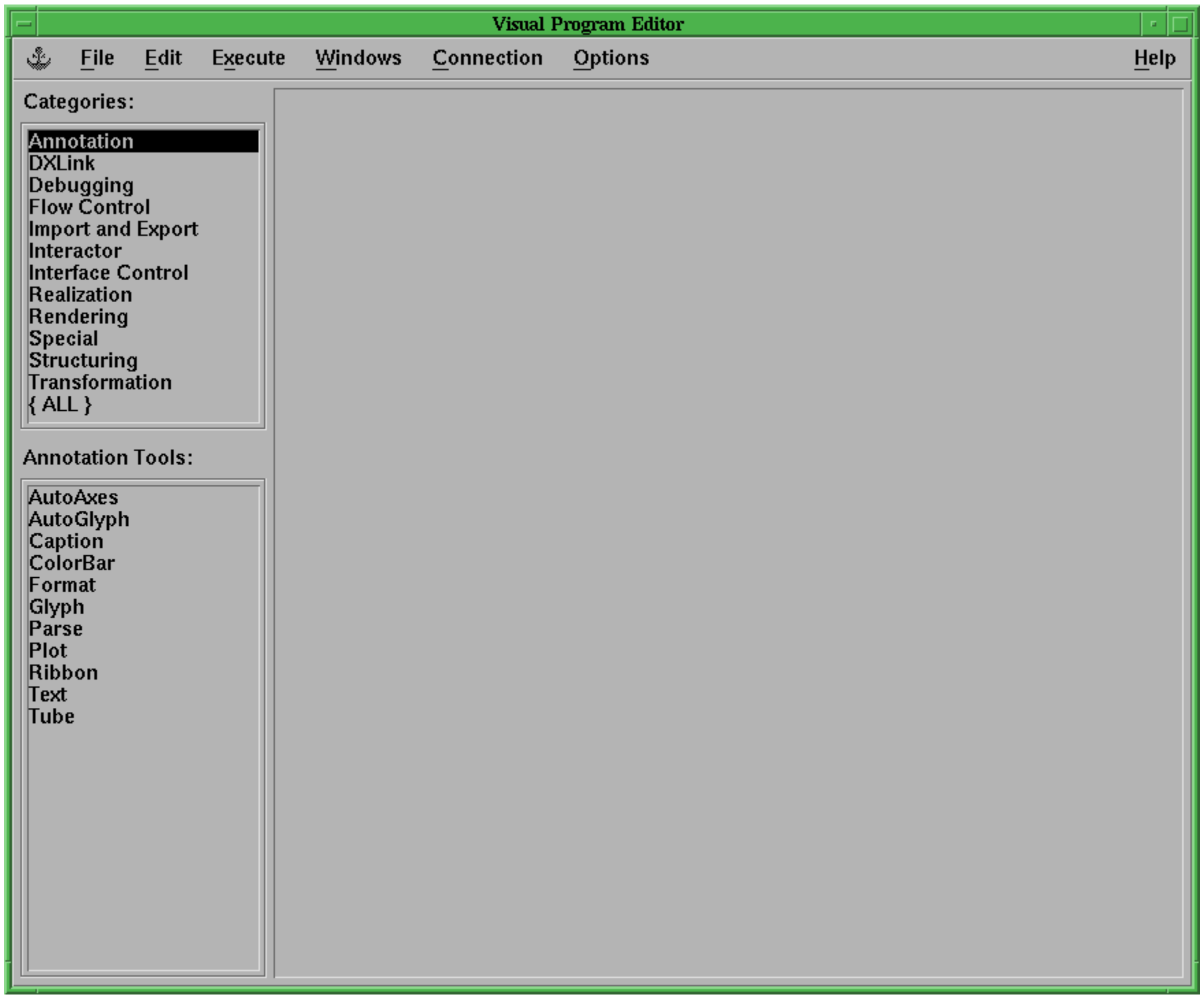


Figure 1: The main window for the Visual Program Editor.

   (b) move the cursor to the new position (continuing to hold down the mouse button),
   (c) release the mouse button.

(5) To delete a module from the canvas:

   (a) click in the body of the module. It should appear highlighted, with a white bar above and below it,
   (b) choose the 'Delete' option from the 'Edit' menu and the module should disappear from the canvas.

(6) The icons representing modules usually have tabs projecting from their top and bottom sides.

   - The tabs along the top represent inputs to the module.
   - The tabs for mandatory inputs are coloured a brighter shade of green; optional (or defaulted) inputs are the same colour as the body of the icon.
   - The tabs along the bottom represent outputs.

   In DX data can only flow from an output tab to an input tab. An input tab can only be fed by one output tab. However, an output tab can feed an arbitrary number of input tabs.

(7) To connect an output to an input:

   (a) click on the output tab and hold the leftmost mouse button down,
   (b) move the cursor to the input tab, continuing to hold the mouse button down,
   (c) release the mouse button.

   A line should be drawn from the output to the input tab (not all input and output types are compatible; if the tabs are not compatible DX will not connect them with a line).

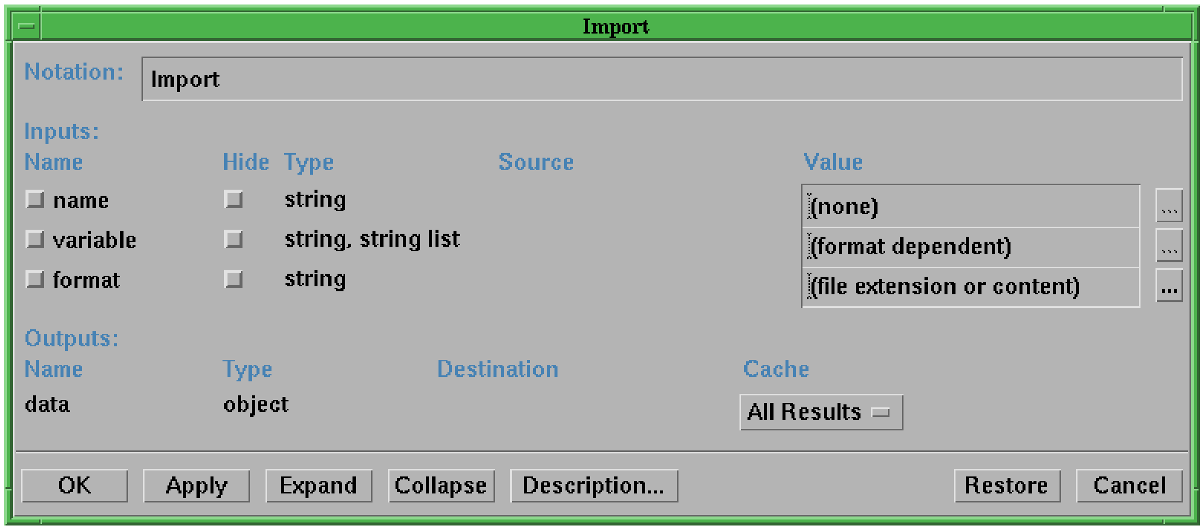
(8) To remove the connection between an input and output:

   (a) click on the *input* tab and hold down the leftmost mouse button,
   (b) move the cursor to a blank portion of the canvas,
   (c) release the mouse button, and the connection will vanish.

(9) At this juncture, practice positioning modules on the canvas, deleting them and connecting them, until you have got the hang of it.

(10) Values for input tabs can be specified as defaults as an alternative to supplying a value from an output tab. This mechanism is often used to specify the name of the data file which a network is to operate on, as well as to supply other values needed by a network. The procedure is as follows:

   (a) double-click on the module for which an input value is to be set,
   (b) a new window, called the configuration window, should appear, showing the commonly used inputs for the module. Figure 2 shows this window for the 'Import' module,



Notation: Import

Inputs:

Name	Hide	Type	Source	Value
name		string		(none)
variable		string, string list		(format dependent)
format		string		(file extension or content)

Outputs:

Name	Type	Destination	Cache
data	object		All Results

Figure 2: Window for the 'Import' module.

(c) each line of the 'Inputs' section of the window corresponds to a separate input; their names are listed on the right hand side. For module 'Import' the inputs are 'name', 'variable' and 'format',

(d) click on the 'Value' box for the chosen input,

(e) hold down the leftmost mouse button and drag it over the existing value, so that the existing value is highlighted,

(f) type in the new value,

(g) hit return (*do not forget this step*),

(h) after you have hit return DX will display the value that you have entered, inside the box, in double quotation marks,

(i) repeat the procedure until all the required inputs have been set, then click on the 'OK' button.

By default only the commonly used inputs for a module are shown. To show all the inputs (and outputs) click on the 'Expand' button.

(11) Various on-line help information is available within DX. In particular, it is possible to get help on individual modules, which is useful both for finding out what a module does and for determining the input which it needs. To obtain help on a module proceed as follows:

(a) click on the 'Help' menu in the top right hand corner of the VPE (see Figure (1)),

(b) choose menu-item 'Context-Sensitive Help'; the mouse pointer will change to a '?',

(c) position the pointer over either the required module on the canvas or the name of the module in the list in the bottom left hand corner of the VPE, and click,

(d) another window containing a detailed description of the module will appear. Note that this information is formatted as hypertext, with links being indicated by a box drawn around the linked text. Simply click inside the box to follow the link,

(e) when you have finished perusing the text simply click on the 'Close' button.

(12) Another useful trick is that if you position the pointer over an input or output tab and hold down the leftmost mouse button then while the button is depressed the name of the tab appears in the module icon. This feature is particularly convenient for identifying tabs.

(13) Now clear the canvas ready to start building a network: select option 'New' from the 'File' menu.

(14) Figure 3 shows virtually the simplest possible DX network. You are now going to create this network.

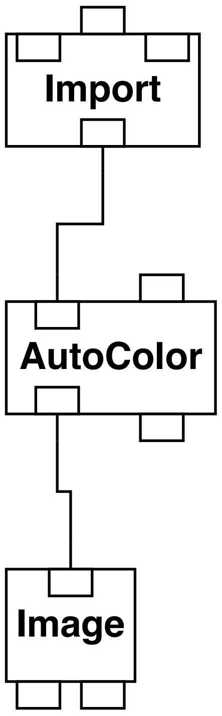


Figure 3: A simple network.

There are two ways to create the network. The first is to find the three modules, position them on the canvas and join them as shown. The second is to read in a prepared copy of the network from a file. For the second option proceed as follows.

(a) Select 'Open Program' from the 'File' menu.

(b) The 'Open. . . ' window appears, as shown in Figure 4. You will use this window to specify the network file.

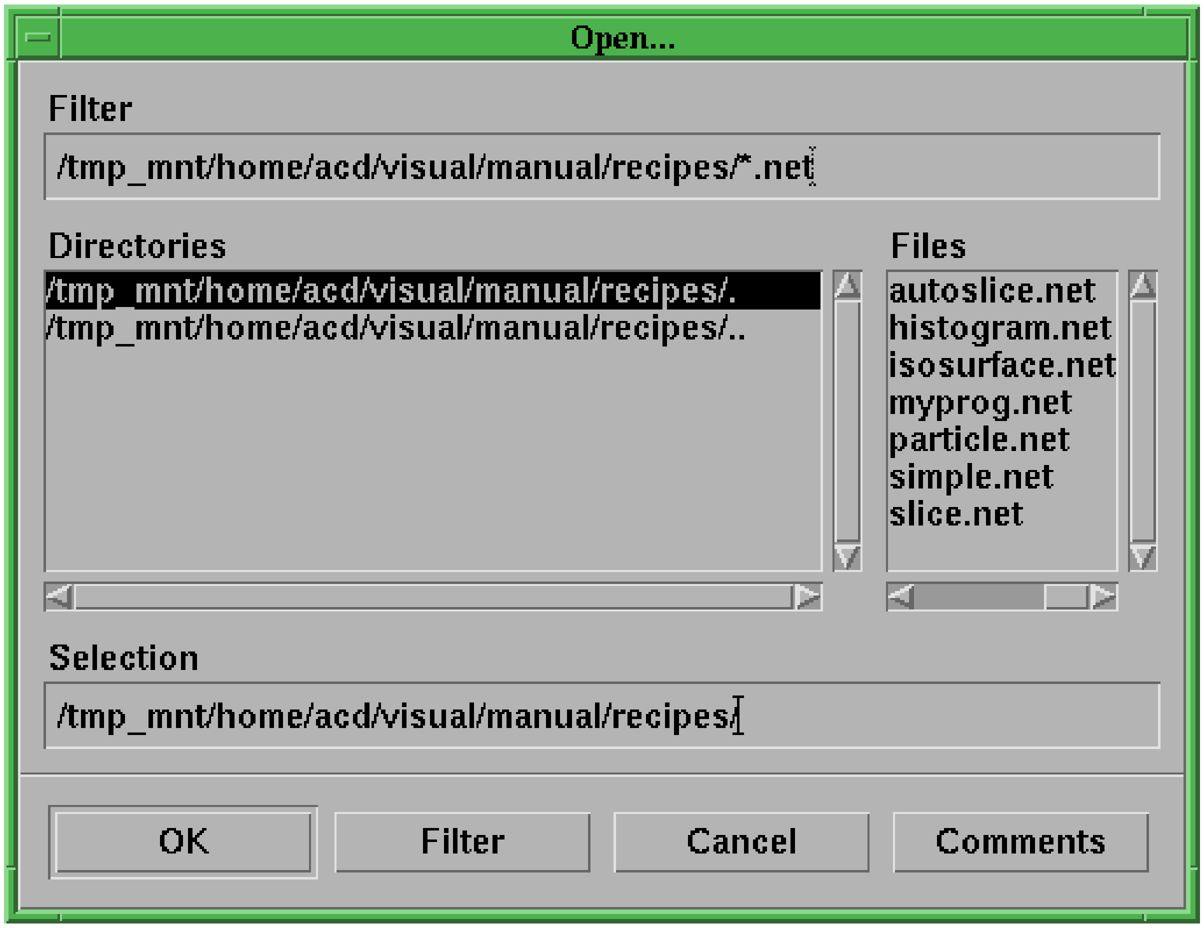

Open...
Filter
/tmp_mnt/home/acd/visual/manual/recipes/*.net
Directories
/tmp_mnt/home/acd/visual/manual/recipes/.
/tmp_mnt/home/acd/visual/manual/recipes/..
Files
autoslice.net
histogram.net
isosurface.net
myprog.net
particle.net
simple.net
slice.net
Selection
/tmp_mnt/home/acd/visual/manual/recipes/
OK
Filter
Cancel
Comments

Figure 4: The 'Open. . . ' window.

(c) The 'Filter' box at the top of the window controls the files available for selection. It contains a wild-card file expression and all the files which match the expression are available for selection. Click on this box and enter:

```
$STARLINK_DIR/examples/sc2
```

A list of networks should appear in the 'Files' box on the right hand side of the window. By convention DX networks have file type '`.net`'.

(d) Click on file '`simple.net`' in this box. The full file name and directory specification of the `simple` network should now appear in the 'Selection' window.

(e) Click on the 'OK' button. The 'Open. . . ' window closes and the network is loaded onto the canvas.

The network is now ready for use.

(15) The purpose of the three modules of the network is:

Import read in the field to be displayed from an external file,

AutoColor colour a field ready for display, so that a range of values in the field map to a range of colours,

Image display a field. In this network 'Image' will produce a volume rendering of a gridded dataset. However, it can accept various other sorts of input to produce different sorts of display.

4.1 Running a network

Running a DX network to generate a visualisation comprises three stages: specifying the input (particularly the file to be read), running the network and examining the result. This example describes how to execute the `simple` network loaded in the previous section. It will be used to view an example data file, supplied with the cookbook, which contains a data cube of a simple Gaussian. This data cube is a 'field' in the context of DX; see Section 17 for an explanation of this term.

(1) Double-click on the 'Import' module; the configuration window shown in Figure 2 should appear.

   (a) Set input 'name' to: `$STARLINK_DIR/examples/sc2/field.general`
   This file is the DX 'header' file describing the data file. Files written by an external program and imported into DX are described by a separate header file. See Sections 5 and 6 for details. Note that it is the name of the header file, *not* the data file, which is given here. If the header file is in your current directory then the directory specification can be omitted.

   (b) Set input 'variable' to: `Gaussian`
   This input specifies which field within the data file is to be read. In the current example the field is called '`Gaussian`'. Again see Sections 5 and 6 for details.

   (c) Set input 'format' to: `general`
   Here '`general`' means that the data file has been written by an external program and is described by a separate header file. The only other option which you are likely to encounter is '`dx`' for a native DX format file.

   (d) When all the values have been set correctly simply click on the 'OK' button.

(2) To run the network select 'Execute Once' from the 'Execute' menu. The network will probably take a few minutes to run.

(3) After a few minutes a window containing the visualisation will appear. The options for this window allow you a great deal of choice in viewing the visualisation. The following couple of hints might be useful.

   - DX is a bit lackadaisical about resetting the limits of the plot when a new visualisation is generated. If your visualisation does not neatly fill the plotting window (it may be either too big or too small) select the 'Reset' option from the 'Options' menu.
   - By default DX generates a view perpendicular to one of the axes, which gives an unnaturally 'flat' appearance to the image. For a better perspective view select the 'View Control' option from the 'Options' menu. The 'View Control' window appears. Set the 'Set View' toggle to 'Diagonal' or one of the 'Off. . . ' options.

5 Importing Gridded Data

It is usually possible to input a file written by some other program into DX. The procedure is simply to create a file which describes the contents of the data file to DX. This description file is called a 'header file' and is simply a text file created with an editor.

As an example this recipe shows how to import a formatted text file containing two data cubes. One data cube is a simple Gaussian field, the other is a Gaussian superimposed on a sloping background. Note that though the values are calculated using Fortran double precision variables they are written out as floating point numbers. DX interprets the ensuing file as containing single precision values. Each data cube is represented as a single DX field (see Section 17). The program to generate the data file is listed in Figure 7.

(1) The DX header file for this data file is shown in Figure 5. These header files have file type '`.general`'. Thus the present example is called '`field.general`'.

```
file=field.lis
format=text
grid=50x50x50
majority=row
interleaving=field
field=Gaussian, Sloping
structure=scalar, scalar
type=float, float
dependency=positions, positions
```

Figure 5: Header file for gridded data.

(2) Each line of the header file consists of a keyword followed by one or more values. The purpose of the various keywords are as follows.

file is the name of the data file. If the file name is not preceded by a directory specification then it is assumed to be in the same directory as the header file.

format is the format of the file; the current example is a formatted file rather than a binary file.

grid is the size of the grid of data. In the present case both data cubes comprise grids with fifty elements in each side.

majority specifies the organisation of multi-dimensional arrays. 'Column' majority means that the first dimension varies fastest (as in Fortran); 'row' majority means that the last dimension varies fastest (as in C).

interleaving specifies how the data for the two fields are interleaved within the file. DX permits various possibilities. For example, all the values for the first field could occur first, followed by all the values for the second field. In the present case the data for the two fields are more closely intertwined. Each record in the file contains a value for a position in the first field and a value for the corresponding position in the second field (see Figures 6 and 7). '`interleaving=field`' specifies this sort of interleaving.

```
0.15655673          0.25655673

0.16572675          0.26572675

0.17514088          0.27514088

0.18476781          0.28476781

0.19457102          0.29457102

0.20450866          0.30450866
    ⋮                   ⋮
```

Figure 6: The first few records of file `field.lis` (written by program `field.f`, Figure 7). The first column is values for the simple Gaussian data cube, the second column values for the Gaussian superimposed on a sloping background.

field specifies names for the fields in the file. Here they are called 'Gaussian' (the simple Gaussian data cube) and 'Sloping' (the Gaussian superimposed on a sloping background). Note the use of a comma to separate the two names.

structure specifies the structure of the two fields in the file. In the current example both the fields are simple scalars.

type specifies the data types of the two fields. In the current example both fields are single precision real numbers.

dependency denotes whether the data are position or connection dependent. See Section 17.4 for an explanation of these two cases. Here both fields are position dependent (which is probably the more common case in astronomy).

A full description of all the possible keywords is given in Section 4.3 *Header File Syntax: Keyword Statements* of the IBM *QuickStart Guide*[1].

(3) Once a suitable description file has been been created the data can be imported into DX by including the 'Import' module in your network. Note that it is the name of the description file, *not* the name of the data file which must be supplied to 'Import'.

5.1 Variations

(1) DX includes 'Data Prompter', a windows-based application for interactively creating the header file for a data set. However, often it is no easier to use than creating the header file with an editor. Section 4.4 *Data Prompter*, of the *QuickStart Guide*[1] gives details.

(2) Unformatted binary files can also be imported using the same header file mechanism. The `format` keyword should be set to '`binary`'. Binary files written with a C program can be imported directly. However, unfortunately, binary files written with a Fortran program contain record-control bytes which must be removed prior to importing the file. The Starlink extensions to DX include `SXUnfort` for this purpose; see SUN/203[11] for details. Of course, an unformatted file written on a Digital alpha will differ from the corresponding file written on a Sun because of the different byte order of the machines. It is possible to input an unformatted file written on a Sun into DX running on a Digital alpha, or vice versa, by using the '`msb`' and '`lsb`' modifiers to the `format` keyword[4]. See Section 4.3 *Header File Syntax: Keyword Statements* of the IBM *QuickStart Guide*[1] for details.

(3) It is also possible to import files in the Starlink NDF format (and other common astronomical formats, such as FITS images or Figaro DST files). Again see SUN/203[11] for details.

[4]Strictly speaking it should be possible to input any unformatted file written in IEEE floating point format, irrespective of the type of machine that it was written on. Note, however, that Digital VAXen and IBM mainframes do not use IEEE floating point format.

```
      PROGRAM FIELD
      IMPLICIT NONE
*    Generate two data cubes.  One contains a Gaussian function, the
*    other a Gaussian on a sloping background.
*    A C Davenhall (Edinburgh), 3/12/95.
      DOUBLE PRECISION
     :  XCEN,   ! X Centre of distribution.
     :  YCEN,   ! Y   "    "       "      .
     :  ZCEN,   ! Z   "    "       "      .
     :  A,      ! Scale factor.
     :  B,      ! Exponential scale factor.
     :  C       ! Background slope.
      PARAMETER
     : (XCEN = 2.5D1,  YCEN = 2.5D1,  ZCEN = 2.5D1,
     :  A = 1.0D1,     B = 1.0D1,     C = 1.0D-1)
      DOUBLE PRECISION
     :  DIST,   ! Distance of point from centre.
     :  VALUE1, ! Value of Gaussian function at the point.
     :  VALUE2  ! Value of Gaussian on sloping background at the point.
      INTEGER
     :  I, J, K ! Loop indices.

      OPEN(UNIT=10, FILE='field.lis', STATUS='NEW')

      DO I = 1, 50
         DO J = 1, 50
            DO K = 1, 50
               DIST = SQRT( ((REAL(I) - XCEN)**2) +
     :           ((REAL(J) - YCEN)**2) +
     :           ((REAL(K) - ZCEN)**2) )

               VALUE1 = A*EXP(-DIST / B)
               VALUE2 = VALUE1 + (C * REAL(I))

               WRITE(10, '(1X, 0PF16.8, 2X, 0PF16.8)' ) VALUE1, VALUE2
            END DO
         END DO
      END DO

      CLOSE(UNIT=10)

      END
```

Figure 7: Program `field.f` to write gridded data.

6 Importing Particle Data

It is usually possible to input a file written by some other program into DX. The procedure is simply to create a file which describes the contents of the data file to DX. This description file is called a 'header file' and is simply a text file created with an editor.

As an example this recipe shows how to input a formatted text file containing particle (or catalogue) data. The positions of the particles trace out a three-dimensional cone. The data are represented as a single DX field (see Section 17). For simplicity the file contains only this single field. However, it is possible to have particle datasets containing several fields, similar to the gridded data in Section 5. The program to generate the data is listed in Figure 9.

(1) The DX header file for this data file is shown in Figure 8. These header files have file type '`.general`'. Thus the present example is called '`particle.general`'.

```
file=particle.lis
points=10000
format=text
field=locations, Intensity
structure=3-vector, scalar
interleaving=field
```

Figure 8: Header file for particle data.

(2) Each line of the header file consists of a keyword followed by one or more values. The purpose of the various keywords are as follows.

file is the name of the data file. If the file name is not preceded by a directory specification then it is assumed to be in the same directory as the header file.

points is the number of particles (or points) in the data set.

format is the format of the file; the current example is a formatted file rather than a binary file.

field specifies names for the the individual fields (or columns) in the data file. In the present case the first three columns, containing the positions are collectively called 'locations' and the final column is called 'Intensity'. Note the use of a comma to separate the two names.

structure specifies the structure of each field in the file. In the present case the first three columns are grouped into a three-element vector containing the positions and the fourth column is treated as a scalar dependent variable.

interleaving specifies how the various data items within the file are intertwined. In the present case each record in the file contains the position and data value for a single particle (see Figure 9). '`interleaving=field`' specifies this sort of interleaving.

A full description of all the possible keywords is given in Section 4.3 *Header File Syntax: Keyword Statements* of the IBM *QuickStart Guide*[1].

(3) Once a suitable description file has been been created the data can be imported into DX by including the 'Import' module in your network. Note that it is the name of the description file, *not* the name of the data file which must be supplied to 'Import'.

6.1 Variations

See the notes for variations of *Importing Gridded Data*, Section 5, above.

```
      PROGRAM PARTICLE
      IMPLICIT NONE
*    Generate a particle dataset, with the individual particles tracing
*    out positions on a cone.  The dependent variable varies
*    (indirectly) with the Z distance.
*    A C Davenhall (Edinburgh), 4/12/95.
      REAL
     :  DR,       ! Increment in radius.
     :  DTHETA,   !     "     "  theta.
     :  DZ,       !     "     "  z.
     :  PI        ! Pi.
      PARAMETER
     : (DR = 1.0E-1,  DTHETA = 1.0E-2,
     :  DZ = 1.0E-1,  PI = 3.1415927E0)
      REAL
     :  R,        ! Radius.
     :  THETA,    ! Theta.
     :  X,        ! X coordinate.
     :  Y,        ! Y     "     .
     :  Z,        ! Z     "     .
     :  VALUE     ! Value of the point.
      INTEGER
     :  I, J, K  ! Loop indices.

      OPEN(UNIT=10, FILE='particle.lis', STATUS='NEW')

      DO I = 1, 10000
         R = REAL(I) * DR
         THETA = REAL(I) * DTHETA
         THETA = MOD(THETA, 2*PI)
         X = R * COS(THETA)
         Y = R * SIN(THETA)
         Z = REAL(I) * DZ

         VALUE = SIN(MOD( REAL(I)/5.0E3, 2*PI) )

         WRITE(10, '(0PE12.4, 1X, 0PE12.4, 1X, 0PE12.4, 1X, 0PE12.4)' )
     :     X, Y, Z, VALUE
      END DO

      CLOSE(UNIT=10)

      END
```

Figure 9: Program `particle.f` to write particle data.

7 Displaying a Histogram

file:	`histogram.net`
example data files:	`field.general`, `particle.general`

This network plots a histogram of the data values in a data set (strictly speaking in a DX field; see Section 17). It also displays the median, minimum and maximum values in the data. It will work on n-dimensional gridded data and n-dimensional particle data. It can be used for preliminary investigation of a dataset about which you have no prior knowledge, in order to determine the range and distribution of values which it contains.

7.1 Use

The network is shown in Figure 10.

- To specify the data cube to be read double-click on the 'Import' module and follow the instructions in Section 4.1.
- By default the histogram will contain 100 bins (unless the data are of data type byte, in which case it will contain 256 bins). To specify the number of bins double-click on the 'Histogram' module and set the input 'bins' to the value required.
- By default the network generates a histogram spanning the entire range of values in the data set. You may need to control the range of values histogramed (for example, to prevent a few rogue values from dominating the x range of the plot). The simplest way to control the x range is to set the 'min' and 'max' inputs of the 'Histogram' module.

7.2 Notes

The histogram is generated by module 'Histogram' and turned into a graph by module 'Plot'. The minimum and maximum values in the data are calculated by 'Statistics'. 'Format' and 'Caption' assemble the median, minimum and maximum into a text string and 'Collect' combines the string with the graph.

7.3 Alternatives

If your data are in Starlink NDF format then an alternative way to generate a histogram is to use command `HIST` in Figaro (see SUN/86[7]). This option will work for n-dimensional gridded data.

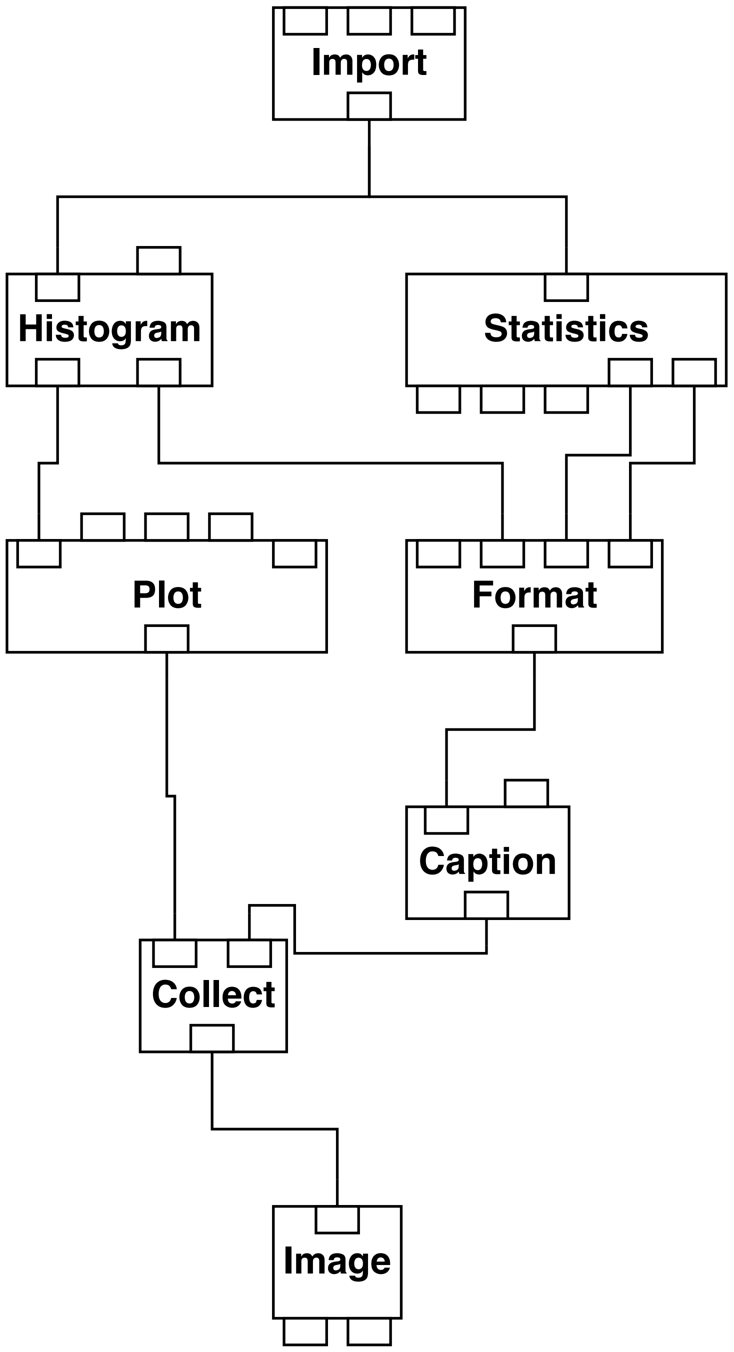


Figure 10: Network to display a histogram.

8 Displaying a Slice

file:	`slice.net`
example data file:	`field.general`

This network displays a two-dimensional slice through a data cube, parallel to two of the axes. The slice is plotted as a false colour array shown in its correct relative position inside a set of axes representing the edges of the data cube. Note that the colour table for converting data values into displayed colours is derived from the range of values in the entire data cube, not just the slice being plotted, in order to facilitate the comparison of several slices.

8.1 Use

The network is shown in Figure 11.

- To specify the data cube to be read double-click on the 'Import' module and follow the instructions in Section 4.1.
- To specify the axis to which the slice is perpendicular double-click on the left-hand of the two 'Integer' modules at the top of the network. The permitted values are 0, 1 and 2 for the x, y and z axes respectively.
- To specify the slice which is to be extracted double-click on the right-hand of the two 'Integer' modules at the top of the network. For a cube with n elements along the chosen axis the permitted values are 0 to $n-1$.

8.2 Notes

Module 'ShowBox' extracts the bounding box of the data cube. 'Slab' extracts the specified slice. 'Statistics' finds the minimum and maximum values in the cube; they are used by 'AutoColor' to generate a false colour image of the extracted slice. 'Collect' combines the bounding box and extracted slice.

8.3 Alternatives

If your data are in Starlink NDF format then an alternative way to plot a slice is to use command `DISPLAY` in KAPPA (see SUN/95[8]). You simply specify the required slice, for example:

```
DISPLAY ndf_file(,,21)
```

plots the twenty-first slice in the third axis.

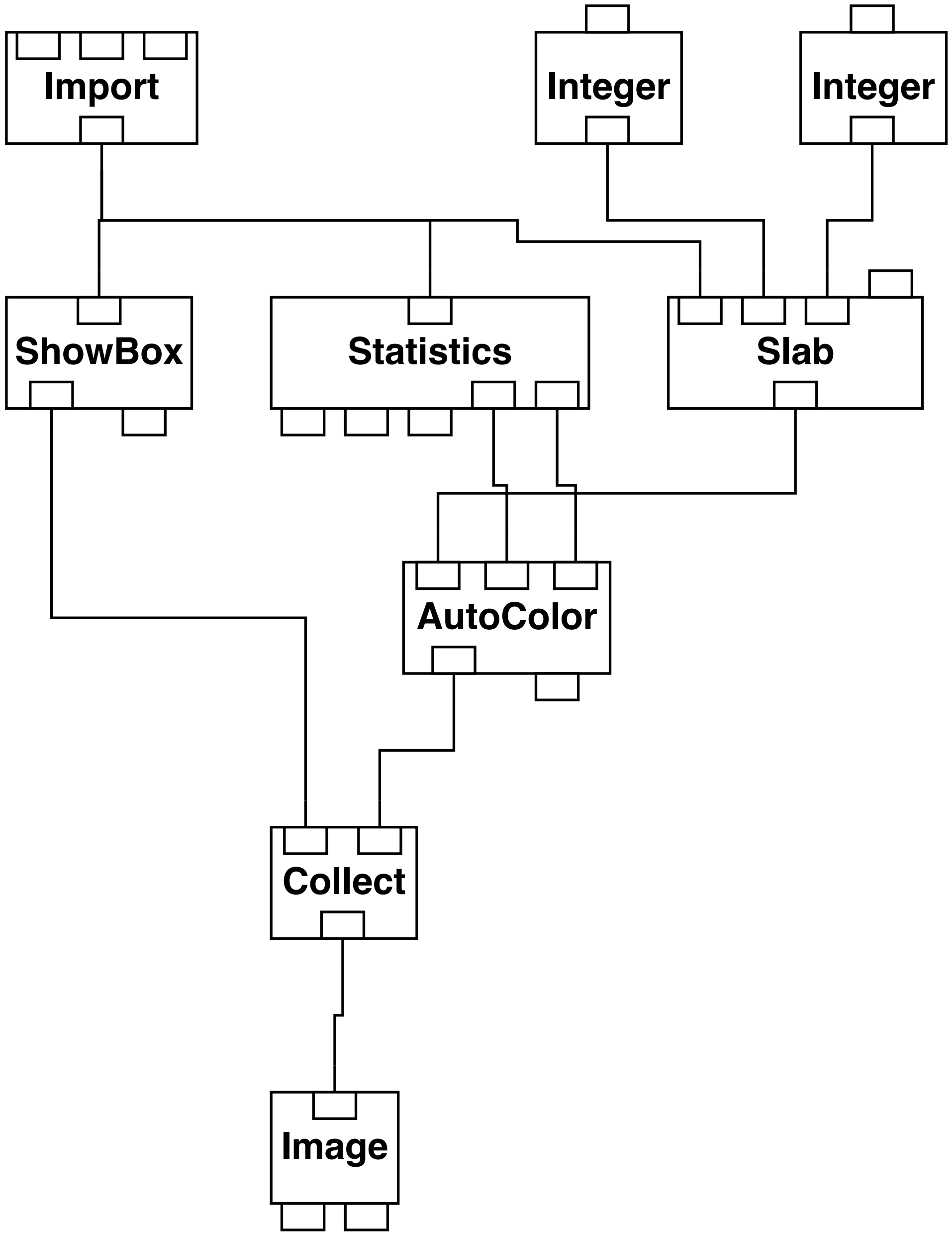


Figure 11: Network to display a slice through a data cube.

9 Displaying a Sequence of Slices

file:	`autoslice.net`
example data file:	`field.general`

This network is a modification of the network for displaying a two-dimensional slice through a data cube parallel to two of the axes (see Section 8). Instead of displaying a single slice it generates an animation showing successive slices 'sweeping' through the data cube. Note that, as before, the colour table for converting data values into displayed colours is derived from the range of values in the entire data cube, not just the current slice being plotted, in order to facilitate visualisation as the current slice sweeps through the data cube.

9.1 Use

The network is shown in Figure 12.

- To specify the data cube to be read double-click on the 'Import' module and follow the instructions in Section 4.1.
- To specify the axis to which the slice is perpendicular double-click on the 'Integer' module at the top of the network. The permitted values are 0, 1 and 2 for the x, y and z axes respectively.
- To specify the range of slices to be shown double-click on the 'Sequencer' module. A control panel with controls similar to (and corresponding functions to) the buttons on a cassette tape deck appears. Click on the ellipsis ('...') button and a further window appears allowing you to specify the first and last slice to be extracted (and the increment between slices). For a cube with n elements along the chosen axis the permitted values are 0 to $n-1$.

  Finally, once you have chosen the required range, click on the play button ('▷') in the original 'Sequencer' window.

9.2 Notes

This network is identical to network `slice.net` for displaying a single slice except that the right-hand of the two 'Integer' modules in `slice.net`, which specified the requested slice, has been replaced with the 'Sequencer' module.

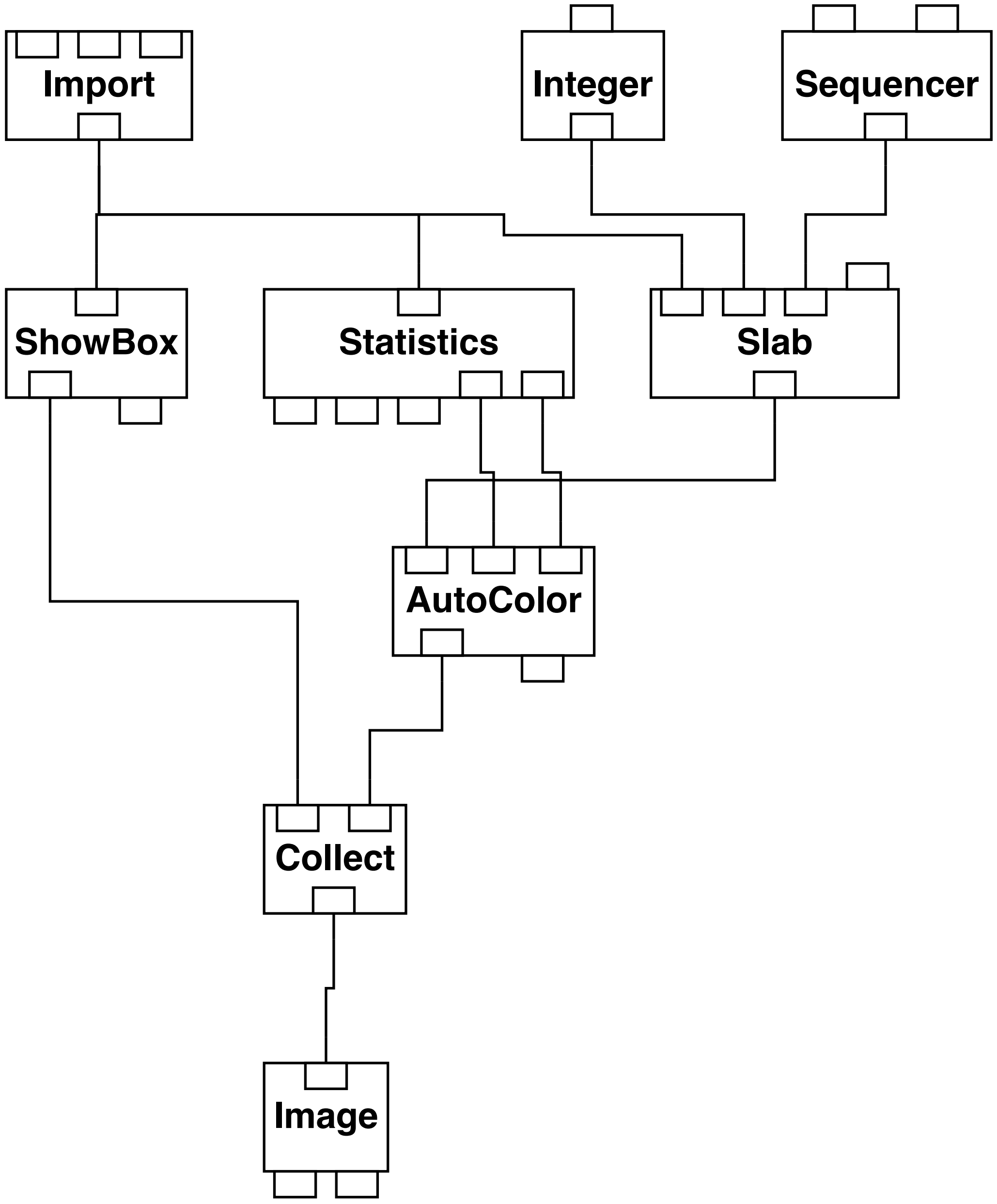


Figure 12: Network to display a sequence of slices in a data cube.

10 Displaying an Iso-surface

file:	`isosurface.net`
example data file:	`field.general`

This network displays an iso-surface[5] corresponding to some value in a data cube. The colour of the iso-surface varies to mimic the effect of light reflecting off the surface. The surface is plotted in its correct relative position inside a set of axes representing the edges of the data cube.

10.1 Use

The network is shown in Figure 13.

- To specify the data cube to be read double-click on the 'Import' module and follow the instructions in Section 4.1.
- To specify the value for the iso-surface double-click on the 'Scalar' module at the top of the network and enter the required value. If you have no knowledge of the range of values in the cube, and hence no idea of suitable value, use network `histogram.net` (see Section 7) to find the minimum, maximum and distribution of values in the data.

10.2 Notes

Module 'ShowBox' extracts the bounding box of the data cube. 'Isosurface' computes the iso-surface and 'AutoColor' colours it. 'Collect' combines the bounding box and iso-surface.

[5]An iso-surface is the analogue in three-dimensional gridded data of a single contour in two-dimensional gridded data. That is, it is a surface defined by some constant value of the dependent variable.

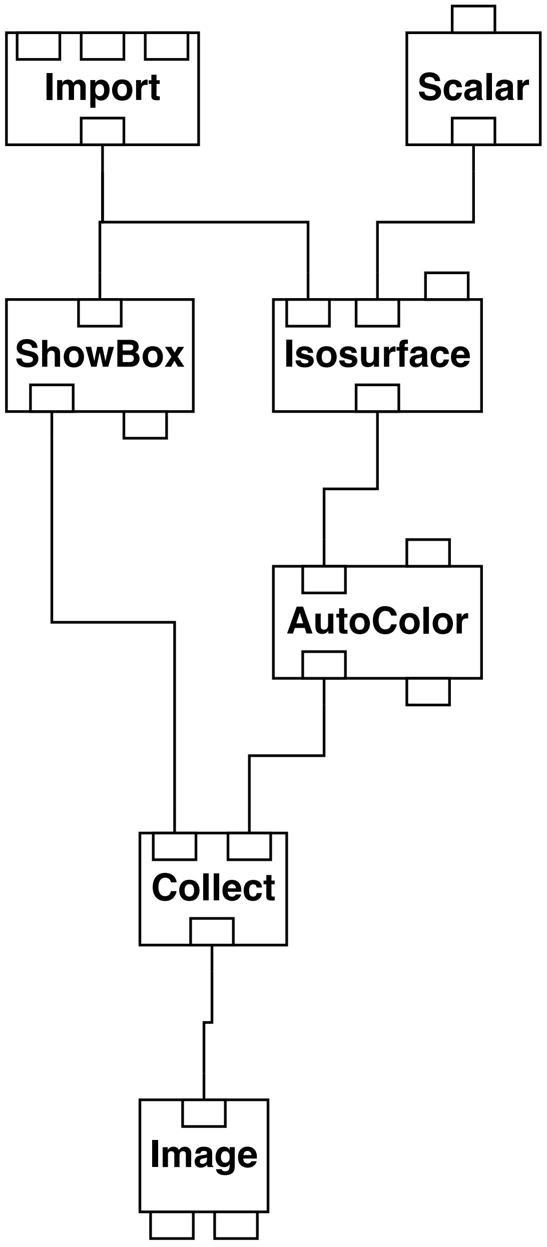


Figure 13: Network to display an iso-surface in a data cube.

11 Displaying Particle Data

file:	`particle.net`
example data file:	`particle.general`

This network displays three-dimensional particle data. Each point is shown in its correct relative position inside a set of axes representing a bounding box just enclosing the dataset. Note however that the position of each point within the plot is necessarily ambiguous. Each point is shown as a dot and coloured according to the data component of the field.

11.1 Use

The network is shown in Figure 14.

- To specify the data set to be read double-click on the 'Import' module and follow the instructions in Section 4.1.

11.2 Notes

Module 'ShowBox' extracts the bounding box of the data set. 'AutoColor' generates a plot showing the positions of each point in the dataset and colours each point according to the value of its data component. 'Collect' combines the bounding box and plot.

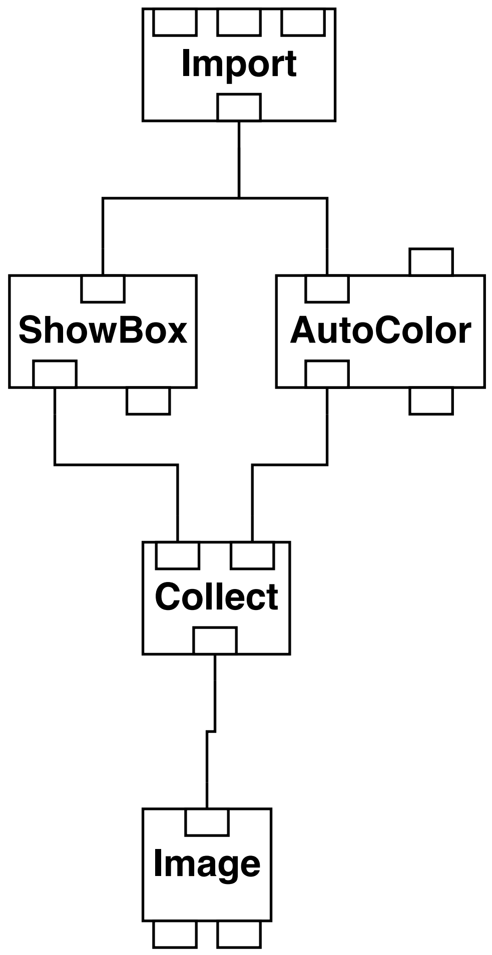


Figure 14: Network to display three-dimensional particle dataset.

Part II

Extended Recipes

12 Introduction

This section presents two extended recipes or case studies of using DX. Both recipes are concerned with importing and visualising three-dimensional 'cubes' of observational data. In one case the data were obtained with the James Clerk Maxwell Telescope (JCMT) in Hawaii and in the other with the ROSAT X-ray astronomy satellite. In both cases two of the axes of the cube correspond to a grid of points on the sky (in some projection) and the third axis to a spectrum in frequency or energy. The grids are regularly sampled in all three axes.

This section assumes that you are familiar with the basics of DX. If you are not familiar with DX then you should probably try some of the simple networks in Part I. At the very least you should be follow the 'Basic Use of DX' recipe (see Section 4).

12.1 Copies of the recipe networks

Like the simple networks in Part I, the networks and data files referred to in this section can be found on Starlink systems in directory:

```
/star/examples/sc2
```

and on non-Starlink systems in directory:

```
$STARLINK_DIR/examples/sc2
```

The most convenient way to access these files is probably to copy them into your current directory before attempting to read them or load them into DX. The examples in subsequent sections assume that copies of the files are available in your current directory.

13 JCMT Data Cube

This recipe shows how to import and display a set of spectra observed in the mm or sub-mm wavelength range of the electromagnetic spectrum with a heterodyne receiver on the James Clerk Maxwell Telescope (JCMT) in Hawaii. The example used here is an observation of emission from the $C^{18}O \quad J = 2 \rightarrow 1$ transition in L1689B, a dense core within the molecular cloud Lynds Dark Nebula 1689. The data cube comprises a grid of 7 by 7 points on the sky, with a spectrum of 153 channels at each position. The example data are available as file `l1689b.sdf`. JCMT heterodyne observations are usually reduced using the SPECX package (see SUN/17[5]) and the example data are in the native SPECX data format. In SPECX terminology a collection of spectra on a grid of points on the sky is a called a 'map'.

The data for L1689B have kindly been made available in advance of publication; they may not be published without the express, written permission of the observers (see Section 18 for details).

(1) The first stage to importing a SPECX map is to convert it to a simple data cube in the standard Starlink NDF format. Application `specx2ndf` in the CONVERT package (see SUN/55[6]) performs this task. Proceed as follows. First type:

```
% convert
```

to make the CONVERT applications available. Then type:

```
% specx2ndf  l1689b  l1689bndf
```

Note that though the input SPECX map and output NDF are respectively held in files called `l1689b.sdf` and `l1689bndf.sdf` they are specified to application `specx2ndf` without the '`.sdf`' file type.

When displaying SPECX maps it is conventional to represent the spectral axis as a series of radial velocities computed for the rest wavelength of the line being observed and corrected to a kinematical local standard of rest. By default `specx2ndf` generates the spectral axis in this fashion and this is how it will be computed in the above example. However, `specx2ndf` allows some flexibility in how the spectral axis is represented; see the appropriate section of SUN/55 for details of the alternatives available.

(2) Once the SPECX map has been converted to a simple data cube in the NDF format it can be processed and displayed with a variety of standard Starlink applications, such as those in the image processing package KAPPA (see SUN/95[8]). For example, type:

```
% kappa
```

to make the KAPPA applications available. Then type:

```
% hislist  l1689bndf
```

Again the file name is specified without the '`.sdf`' file type. This application lists the history information describing the processing which has been performed on the data. The history information appended by `specx2ndf` includes details of the way in which the spectral axis was computed. It is also possible to inspect the structure of the data set using `hdstrace`. Simply type:

```
% hdstrace  l1689bndf
```

As usual the file name is specified without the '.sdf' file type. This facility is useful because it lists the value of much of the auxiliary information contained in the data set. hdstrace is documented in SUN/102[10]. In addition hdstrace will also work on the original SPECX map.

(3) The next step is to convert the NDF format file into a file in the 'native' DX format, which DX can read. This operation could be performed 'on the fly' as DX reads the file. However, it is simpler to convert the file in a separate operation prior to invoking DX. The conversion is performed using application ndf2dx, which is part of SX, the Starlink enhancements to DX (see SUN/203[11]). By convention files in the native DX format have file type '.dx'. To convert the entire data cube simply type:

```
% $SX_DIR/ndf2dx  l1689bndf  l1689ball.dx
```

Note that though the file type is not specified for the input NDF file, it should be given for the output native DX format file.

The above example will convert the entire data cube. However, often the useful information will be contained in only a small range of radial velocities. For example, in the example map most of the useful information lies between velocity channels 65 and 85. It is possible to convert a subset of the NDF corresponding to a given range of velocity channels. For example, to convert a subset corresponding to channels 65 through to 85 type:

```
% $SX_DIR/ndf2dx  'l1689bndf(65:85,,)'  l1689bsub.dx
```

The syntax to specify a subset of an NDF is to give the bounds of the required region inside parentheses after the file name. Unfortunately however, by default the Unix shell will attempt to interpret these parentheses. Thus, in the above example the input file name and NDF subset are enclosed in single quotes in order to prevent this behaviour and ensure they are passed correctly to ndf2dx. The use of 'escape mechanisms' of this sort to prevent the premature interpretation of special characters sent to Starlink applications is discussed in SC/4[3].

(4) Figure 15 shows a DX network to display a sequence of slices through a JCMT map. Each slice corresponds to the grid of points seen on the sky at a given radial velocity. The network is available as files jcmtslice.net and jcmtslice.cfg (jcmtslice.net is the basic network and jcmtslice.cfg is a 'configuration file' which controls some aspects of its behaviour). Start DX (as described in Section 2). Then proceed as follows.

   (a) Load the network. Select the 'Open Program' option from the 'File' menu of the main DX window. Select file jcmtslice.net. The network should now load and appear in the main window. See Section 4 for further details.
   (b) Select the 'Open All Control Panels' option from the 'Windows' menu of the main DX window. Set the required file name and number of slices. (In the current example these are l1689bsub.dx and 20 respectively.) Close the 'Control Panel' window.
   (c) Execute the network once. Once the network has executed you will probably have to reset the display window (option 'Reset' in the 'Options' menu of the display window). For a more effective display select 'View control' from the 'Options' menu and set the 'Set View' parameter to one of the 'Off' options, perhaps 'Off top'.

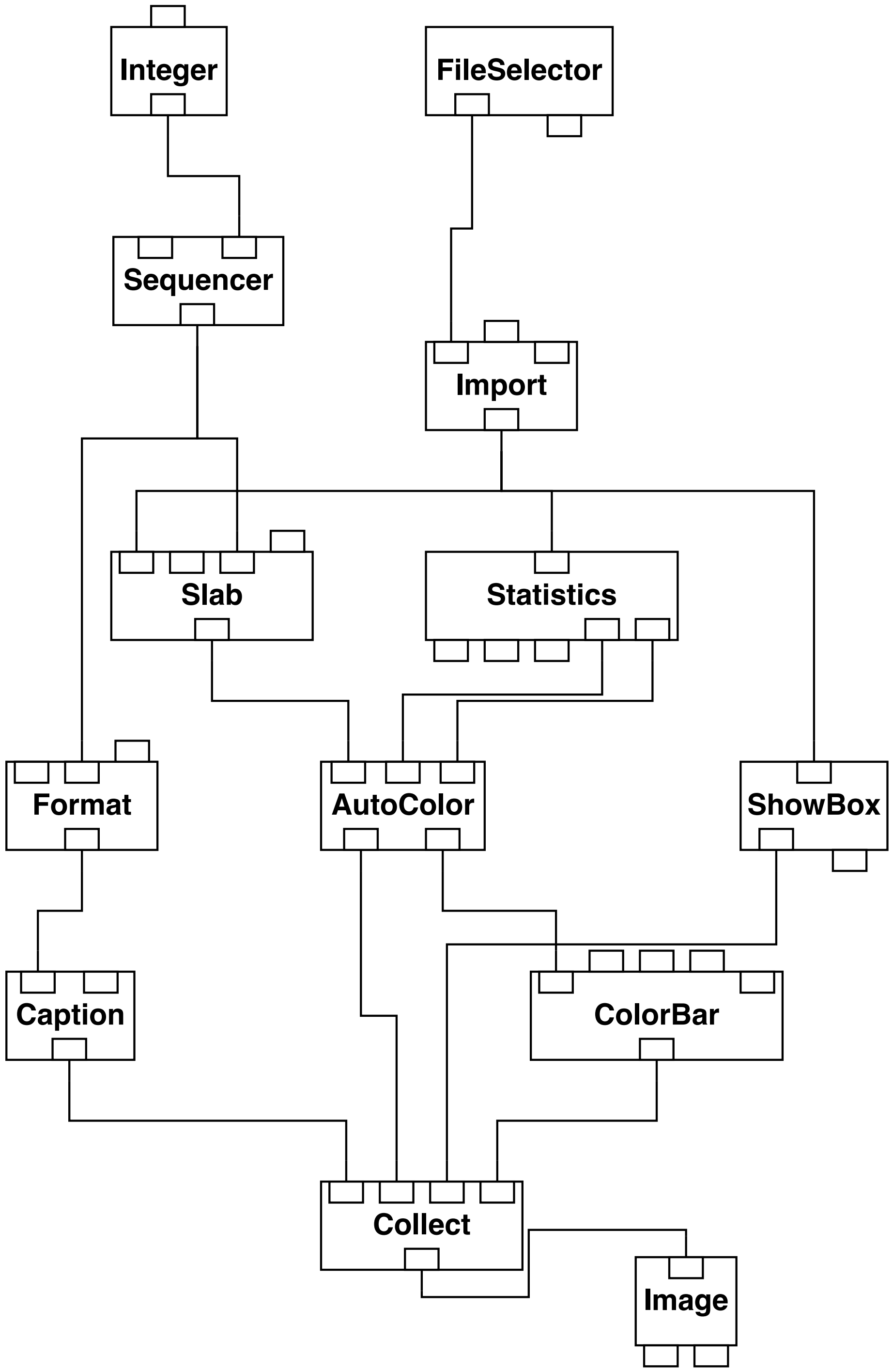


Figure 15: Network to display a sequence of slices through a JCMT data cube.

   (d) Finally double click on the 'Sequencer' module and click on the play button. A sequence of slices sweeping through the data cube will now be displayed.

   The network includes a 'scale bar' showing how the colour displayed for each pixel in each slice corresponds to the value of the pixel. In this example the value of each pixel is the radio 'brightness temperature' in kelvin.

(5) Figure 16 shows a network for displaying an iso-surface[6] in a JCMT data cube. This network is available as files `jcmtsurface.net` and `jcmtsurface.cfg` (again `jcmtsurface.net` is the basic network and `jcmtsurface.cfg` is a configuration file).

   Much of the complexity of this network occurs because it contains an option to normalise the axes of the cube. Two of the axes correspond to positions on the sky and the third to the dispersion, usually expressed as a radial velocity computed from the rest wavelength of the line. There is no relation between the spacial and spectral axes, so the 'cube' can be either very 'long' or 'thin' in the spectral axis compared to the spacial ones. In order to generate an effective iso-surface it often helps to 'normalise' the cube so that it has unit length in each axis. Though this procedure often improves the iso-surfaces generated it makes the axes less intuitive. Hence it is made optional so that you can choose the alternative most suitable for your data.

   (a) Select the 'Open All Control Panels' option from the 'Windows' menu of the main DX window. Set the axis units, file name and iso-surface value. In the current example suitable values of these quantities are: normalised, `l1689bsub.dx` and 3.5 respectively. The contour (or iso-surface) value is specified as a radio 'brightness temperature' in kelvin. Close the 'Control Panel' window.

   (b) Execute the network once. Once the network has executed you will probably have to reset the display window (option 'Reset' in the 'Options' menu of the display window). For a more effective display select 'View control' from the 'Options' menu and set the 'Set View' parameter to one of the 'Off' options, perhaps 'Off top'. Re-execute the network.

   (c) Use the various settings of the 'Mode' option in the 'View Control' window to pan, zoom and rotate the display in order to examine the iso-surface. These options are described in Section 2.3 *Controlling the Appearance of an Object: the Image Window* of the IBM *QuickStart Guide*[1].

[6]An iso-surface is the analogue in three-dimensional gridded data of a single contour in two-dimensional gridded data. That is, it is a surface defined by some constant value of the dependent variable.

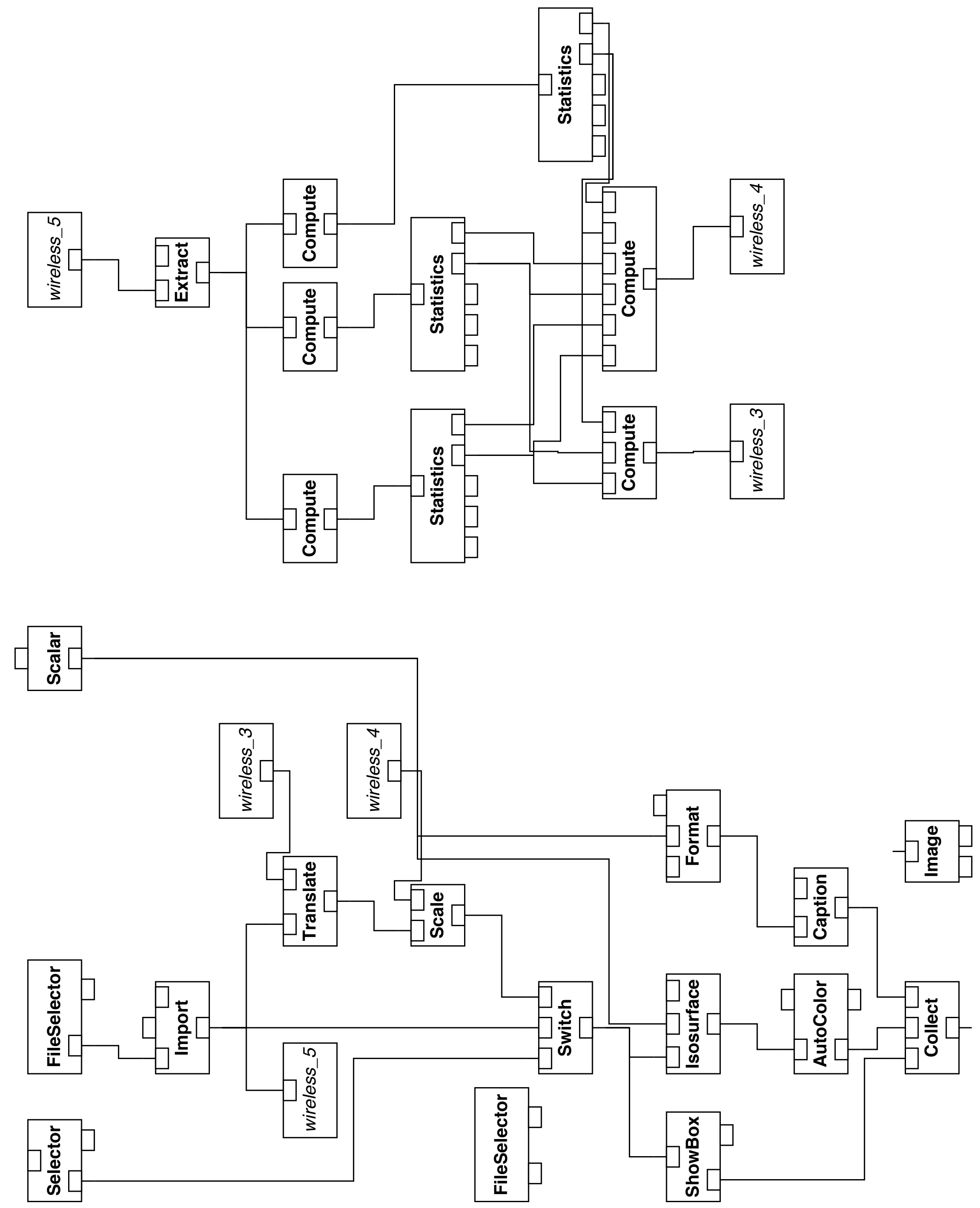


Figure 16: Network to display an iso-surface in a JCMT data cube.

14 ROSAT Data Cube

This recipe shows how to import and display a data cube observed with the ROSAT X-ray astronomy satellite (*Röntgensatellit*). ROSAT data come in a number of different formats. The example used here is an Asterix binned dataset of the irregular galaxy M82 (NGC 3034). This format is used by the Asterix package (see SUN/98[9]). The data cube comprises a grid of 216 by 216 points on the sky, with a spectrum of 22 points at each position. The example data are available as file `m82ast.sdf`. The procedure for importing and displaying these data is very similar to the corresponding procedure for JCMT data described in Section 13.

(1) The first stage to importing an Asterix binned dataset is to convert it to a simple data cube in the standard Starlink NDF format. Application `ast2ndf` in the CONVERT package (see SUN/55[6]) performs this task. Proceed as follows. First type:

```
% convert
```

to make the CONVERT applications available. Then type:

```
% ast2ndf   m82ast  m82ndf
```

Note that though the input Asterix binned dataset and output NDF are respectively held in files called `m82ast.sdf` and `m82ndf.sdf` they are specified to application `ast2ndf` without the '`.sdf`' file type.

It is also worth noting that an Asterix binned dataset is itself almost a standard Starlink NDF. `ast2ndf` merely makes the data format completely standard and rearranges the order of the axes.

(2) Once the Asterix binned dataset has been converted to a simple data cube in the NDF format then a variety of standard Starlink applications can be used to process and display it. For example, it can be accessed with the image processing applications in KAPPA (see SUN/95[8]). It is also possible to inspect the structure of the data set using `hdstrace`. Simply type:

```
% hdstrace  m82ndf
```

Again the file name is specified without the '`.sdf`' file type. This facility is useful because it lists the value of much of the auxiliary information contained in the data set. `hdstrace` is documented in SUN/102[10]. In addition, `hdstrace` and many other standard Starlink applications, will also work on the original Asterix binned dataset.

(3) The next step is to convert the NDF format file into a file in the 'native' DX format, which DX can read. This operation could be performed 'on the fly' as DX reads the file. However, it is simpler to convert the file in a separate operation prior to invoking DX. The conversion is performed using application `ndf2dx`, which is part of SX, the Starlink enhancements to DX (see SUN/203[11]). By convention files in the native DX format have file type '`.dx`'. To convert the entire data cube simply type:

```
% $SX_DIR/ndf2dx  m82ndf  m82.dx
```

Note that though the file type is not specified for the input NDF file, it should be given for the output native DX format file.

The above example will convert the entire data cube. However, often the useful information will be contained in only a small range of energies. For example, in the example dataset most of the useful information lies between energy steps 5 and 15. It is possible to convert a subset of the NDF corresponding to a given range of energy steps. For example, to convert a subset corresponding to steps 5 through to 15 type:

```
% $SX_DIR/ndf2dx  'm82ndf(5:15,,)'  m82sub.dx  axes=no
```

The syntax to specify a subset of an NDF is to give the bounds of the required region inside parentheses after the file name. Unfortunately however, by default the Unix shell will attempt to interpret these parentheses. Thus, in the above example the input file name and NDF subset are enclosed in single quotes in order to prevent this behaviour and ensure they are passed correctly to `ndf2dx`. The use of 'escape mechanisms' of this sort to prevent the premature interpretation of special characters sent to Starlink applications is discussed in SC/4[3].

The `axes=no` option causes `ndf2dx` to ignore any axis information present in the input dataset and write the output DX file with axes consisting of simple pixel numbers. This option may or may not be appropriate depending on the details of your data. In the present case it leads to a better visualisation.

(4) Figure 17 shows a DX network to display a sequence of slices through a ROSAT data cube Each slice corresponds to the grid of points seen on the sky at a given radial velocity. The network is available as files `rosatslice.net` and `rosatslice.cfg` (`rosatslice.net` is the basic network and `rosatslice.cfg` is a 'configuration file' which controls some aspects of its behaviour). Start DX (as described in Section 2). Then proceed as follows.

  (a) Load the network. Select the 'Open Program' option from the 'File' menu of the main DX window. Select file `rosatslice.net`. The network should now load and appear in the main window. See Section 4 for further details.
  (b) Select the 'Open All Control Panels' option from the 'Windows' menu of the main DX window. Set the required file name and number of slices. (In the current example these are `m82sub.dx` and 10 respectively.) Close the 'Control Panel' window.
  (c) Execute the network once. Once the network has executed you will probably have to reset the display window (option 'Reset' in the 'Options' menu of the display window). For a more effective display select 'View control' from the 'Options' menu and set the 'Set View' parameter to one of the 'Off' options, perhaps 'Off top'.
  (d) Finally double click on the 'Sequencer' module and click on the play button. A sequence of slices sweeping through the data cube will now be displayed.

The network includes a 'scale bar' showing how the colour displayed for each pixel in each slice corresponds to the value of the pixel. In this example the value of each pixel is the count rate in counts per second. This network for displaying ROSAT data is virtually identical to the corresponding network for JCMT data (see Section 13 and Figure 15). The only differences are the defaults for the file names and the axis labels. Similarly, the network for generating an iso-surface in JCMT data (see Figure 16) requires only minor cosmetic modifications to display ROSAT data.

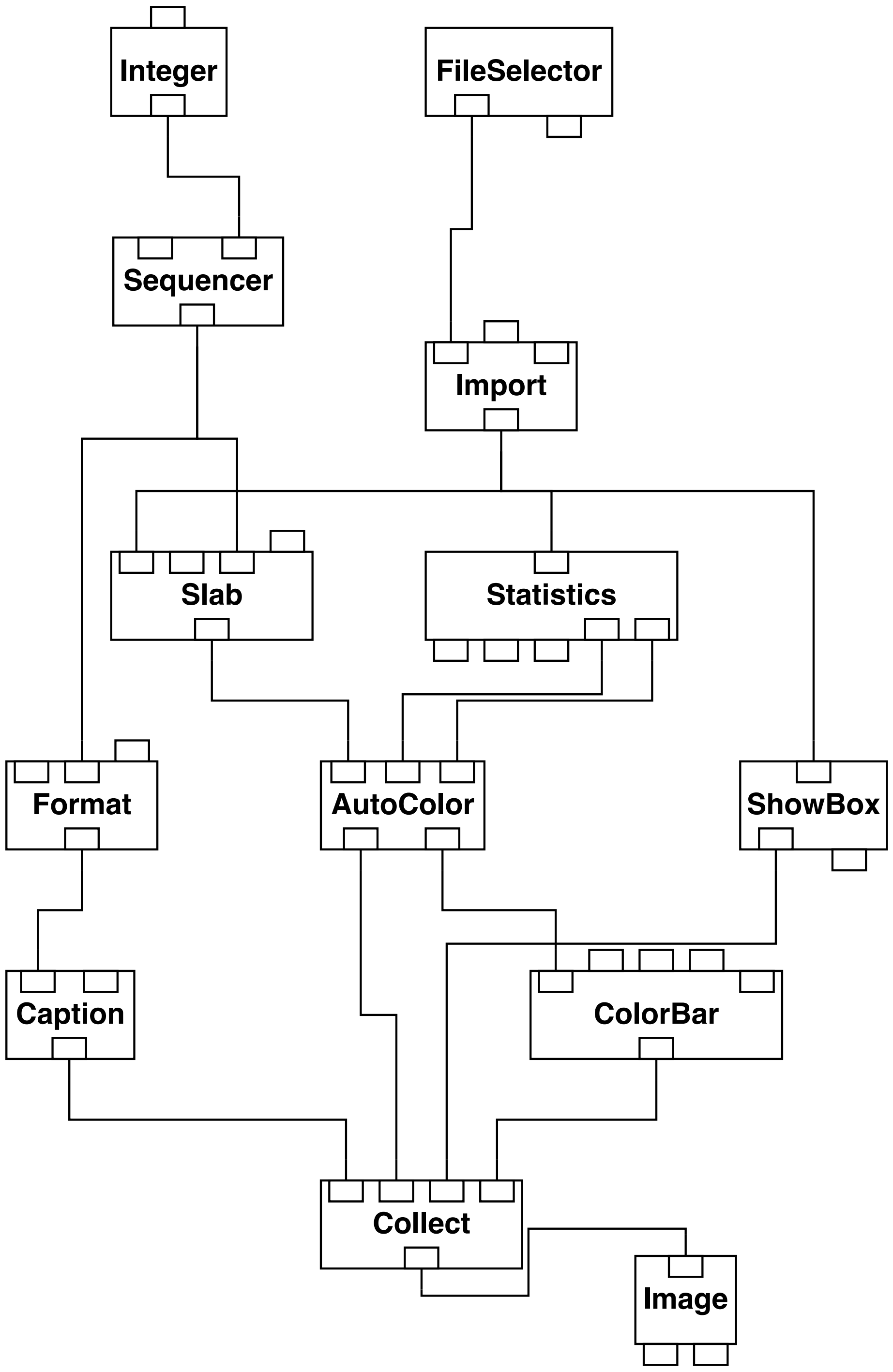


Figure 17: Network to display a sequence of slices through a ROSAT data cube.

Part III

The DX Data Model

15 Introduction

DX has a powerful and flexible data model capable of representing many sorts of data. Whilst you do not need to understand the data model in detail, you will be able to use DX more effectively if you have a rudimentary understanding of it. Section 16 gives a very brief summary of the data model. If you do not like reading manuals it might just about give you enough information to get by. Section 17 presents a slightly longer, but still quite short, introduction to the data model. I recommend that you read it if you are planning to make a significant amount of use of DX.

15.1 Further reading

The DX data model is fully described in Chapter 2 *Introduction to Visualization*, especially Section 2.1, and Chapter 3 *Understanding the Data Model* of the IBM *User's Guide*[2].

16 Summary

In DX the basic entity used to represent a data cube or a particle (or catalogue) dataset is called a **field** (this usage is similar to the use of the term 'field' in physics, for example, 'velocity field', but is different from the usual use in computing). Each field consists of a set of **components**; arrays holding the **positions** of the data points and a **data** array containing the dependent variable. The data array may contain either scalar or vector data. For particle (or catalogue) data, where the positions of the points are not related to each other, the field need only contain the positions and data components. However, for gridded data there is an additional **connections** array which defines the relation between neighbouring points in the grid and controls how interpolation is performed.

In DX gridded data may be either position dependent or connection dependent.

Position dependent The data array contains instantaneous samples of the underlying field computed at the position of each grid point.

Connection dependent The grid points delineate the corners of a cell. The data array contains the value inside this cell.

The practical difference between the two cases is that for position dependent data new values for positions intermediate between grid points are interpolated, whereas for connection dependent data they are not (the single data value applies to the entire cell). Position dependent data are probably more common in astronomy.

17 Outline of the Data Model

The data structures which can be represented using the DX data model include:

(1) data defined on a regular orthogonal grid,

(2) data defined on a deformed regular or curvilinear grid,

(3) data defined on various sorts of irregular grids,

(4) unstructured data with no connection between the data points.

In the first case the dependent variable (which is the quantity to be visualised), perhaps for example, temperature, is sampled in a regular grid inside some region defined with the independent variables as axes. Usually the independent variables will be positions in two, three or higher dimensions. Typically inside your own programs such data would be represented as an array of appropriate dimensionality. The second and third cases are generalisations of the first and can be ignored for our present purposes. The fourth case corresponds to particle (or catalogue) data. Here each point is simply a point in an assemblage or cloud of points; there is no connection in terms of either the independent variables (positions) or the dependent variable between the points. An example might be a simulation of a globular cluster where each data point corresponds to a separate star following its own orbit inside the cluster.

Any of the four cases can occur with any dimensionality: one, two, three and higher dimensional data can be represented. Similarly all the usual data types are available: integer, real, double precision, complex etc. The dependent quantity may be a scalar (such as temperature, pressure or energy) or a vector (such as velocity or momentum).

The fundamental entity in the DX data model is an **object**. Data are represented by the same set of objects both when they are resident in memory when DX is running and when they are stored in native format disk files[7]. There are several sorts of objects. For the present purposes the most important type of object is the **field**; usually each separate dataset will be represented as a single field[8].

A field consists of an arbitrary number of **components** and each component itself has a number of **attributes**. This hierarchy is illustrated in Figure 18.

17.1 Components

There are many different types of component. Some are mandatory, but most are optional. Only a few of the more common ones are relevant to the present discussion. They are listed in Table 1 and described below.

[7]It is also, of course, possible to input files written in a wide variety of other formats; see Sections 5 and 6 for some examples.

[8]This usage of the term 'field' corresponds to its usual meaning in physics rather than in computing. For example, a DX field may represent a three-dimensional grid holding a set of samples of the velocity field throughout some volume. This usage is quite different from the usual meaning in computing: a predetermined section of a record allocated to the storage of a particular data item. Anyone used to the computing terminology should particularly beware of this different usage.

```
field
  component 1
    attribute 1
    attribute 2
          .
          .

  component 2
    attribute 1
    attribute 2
          .
          .

  component 3
        .
        .
```

Figure 18: The hierarchy of items in a DX field. Indenting to the right indicates successively lower levels in the hierarchy.

Name	Mandatory?	Comments
positions	●	position of each datum
data	●	value of each datum
connections		relation between data points
box		bounding box for positions in the field

Table 1: Common components of a field.

Positions The position component is an array[9] defining a position for every datum in the dataset. If the field contains unconnected particle data then the array will be a simple list with a position for each datum. If the field contains a regular grid the positions can be represented more compactly as a **regular** or **product** array; essentially just the origin, size and increment of the grid are stored.

Data The data component is an array storing the dependent variable: temperature, density, velocity, momentum or whatever.

Connections The connections component prescribes to DX how to perform interpolation between neighbouring grid elements. It is described further in Section 17.3, below. In unstructured particle or catalogue data the connection component is absent (because the position of one particle bears no relation to the position of any other particle; think of the example of stars following their individual orbits within a globular cluster).

Box The box component is an array of 2^n points where n is the dimensionality of the positions component. The values contain the coordinates of a bounding box sufficiently large to just enclose all the positions of a field.

17.2 Attributes

Components can have **attributes** associated with them. Attributes are additional items of information describing some aspect of the component. For example, the dependency attribute, 'dep', identifies another component on which the component depends; the 'data' component may depend on either the positions or the connections component, depending on circumstances. Alternatively the component 'ref' indicates that a component refers to another component. Typically the connections component will have a 'ref' of 'positions', indicating that the connections refer to the positions.

17.3 The connections component and interpolation

The connections component prescribes to DX how to perform interpolation between neighbouring grid elements. It describes the logical connectivity between adjacent neighbours in a grid. For example, the interpolation could be performed linearly along one of the axes, or bi- or tri-linearly along two or more axes simultaneously.

The connections component is implemented as an array. Each point in the positions component is given an ordinal number (starting at zero). The connections array comprises a 'list of lists' of numbers in which each entry represents the ordinal values of the points that are to be connected, listed in the order that they are to be connected.

You will not normally need to be concerned with the full details of the connections component. For a regular grid it can be created automatically when the data are imported into DX and represented compactly using a **path** or **mesh** array.

Fields representing particle or catalogue data where there are no logical connections between elements in the positions component will not usually have a connections component (again think of the example where each point represents a star orbiting in a globular cluster).

[9]Strictly speaking this array, and other arrays in components, are themselves DX objects of type **array**. However, this complication is not important to the present discussion and it is adequate to think of the component as simply an array of numbers.

17.4 Position and connection dependent data

For a field of gridded data (it is easier to think of a regular grid, though the argument applies equally to an irregular grid) there are two ways in which the dependent variable can relate to the elements of the grid.

Position dependent The dependent variable represents instantaneous samples of the underlying field (temperature, pressure, momentum or whatever), computed at the position of each grid point.

Connection dependent The grid points delineate the corners of a cell. The dependent variable is the value inside this cell. An example might be a grid spanning a globular cluster with the dependent variable being the number of stars in each cell.

The practical difference between the two cases is the way in which values for the dependent variable are computed at positions intermediate between grid points.

Position dependent A value for the dependent variable is interpolated from its values at the neighbouring grid points.

Connection dependent The value of the dependent variable for the cell is used for any positions inside the cell (because the value pertains to the entire cell, not just one instantaneous point).

The dependency attribute, 'dep', of the data component will be set to 'positions' if the data are position dependent and to 'connections' if they are connection dependent.

Before you input data to DX you must decide whether they are position dependent or connection dependent. Position dependent data are probably more common in astronomy.

18 Acknowledgements

Derek Ward-Thompson and Nick Jessop kindly provided the JCMT SPECX map and Dave Strickland the ROSAT Asterix binned dataset. David Berry gave much useful advice and assistance with DX and suggested several improvements to a draft version of the document.

References

The references are listed alphabetically by title. In the case of Starlink documents the document code (for example SC/4.1 or SUN/17.6) forms part of the title.

[1] *IBM Visualization Data Explorer: QuickStart Guide*, Second Edition, 1995, document reference number: SC34-3262-01. 1, 3.2, 4, (2), (1), (2), (2), (5)c

[2] *IBM Visualization Data Explorer: User's Guide*, Sixth Edition, 1995, document reference number: SC38-0496-05. 15.1

[3] SC/4.1: *C-shell Cookbook* by M.J. Currie, 2 May 1997 (Starlink). (3), (3)

[4] SG/8.2: *An Introduction to Visualisation Software for Astronomy* by A.C. Davenhall, 4 March 1997 (Starlink). 1

[5] SUN/17.6: *SPECX – A Millimetre Wave Spectral Reduction Package* by R.M. Prestage, H. Meyerdierks and J.F. Lightfoot, 13 January 1995 (Starlink). 13

[6] SUN/55.6: *CONVERT – A Format-conversion Package* by M.J. Currie, G.J. Privett and A.J. Chipperfield, 7 December 1995 (Starlink). (1), (1)

[7] SUN/86.12: *FIGARO – A General Data Reduction System* by K. Shortridge, H. Meyerdierks, M.J. Currie and M.J. Clayton, 26 September 1996 (Starlink). 7.3

[8] SUN/95.9: *KAPPA – Kernel Application Package* by M.J. Currie, 9 December 1995 (Starlink). 8.3, (2), (2)

[9] SUN/98.6: *ASTERIX – X-ray Data Processing System* D.J. Allan and R.J. Vallance, 13 May 1995 (Starlink). 14

[10] SUN/102.3: *HDSTRACE – Listing HDS Data Files* by M.J. Currie, 14 February 1994 (Starlink). (2), (2)

[11] SUN/203.3: *DX – IBM Data Explorer for Data Visualisation* by D.S. Berry, G.J. Privett and A.C. Davenhall, 15 September 1997 (Starlink). 1, 4, (2), (3), (3), (3)