CCLRC / Rutherford Appleton Laboratory
Particle Physics & Astronomy Research Council
Starlink Project
Starlink User Note 203.3

SUN/203.3

D.S. Berry, G.J. Privett and A.C. Davenhall
15 September 1997

SX & DX — IBM Data Explorer for Data Visualisation

Abstract

This manual documents the use of IBM DX (Data Explorer) on Starlink systems. DX is a commercial scientific data visualisation package, suitable for the visualisation and display of many sorts of astronomical data. It is the package which Starlink recommends for the display of three-dimensional scalar and vector data. DX is not available at all Starlink sites; your site manager should be able to advise on whether or not it is available at your site.

This manual describes how to access DX and SX, the Starlink enhancements to DX. It also documents these enhancements.

This edition of the manual applies to version 3.1 of DX and describes version 1.1 of SX. If you have a later version of DX it may be necessary to re-build SX. Your site manager should be able to advise on this.

Contents

List of Tables

Revision history

1. 31st January 1996: Version 1 (DSB, GJP, ACD).
2. 3rd March 1997: Version 2. Minor changes (DSB, ACD).
3. 15th September 1997: Version 3. Revised the documentation for `ndf2dx` to include the new parameter `axes` (ACD).

1 Introduction

This manual is a guide to the use of IBM DX (Data Explorer) on Starlink systems. DX is a scientific visualisation package, suitable for the visualisation and display of many sorts of astronomical data. It is the package which Starlink recommends for the display of three-dimensional scalar and vector data. It has particularly good features for the display of three-dimensional vector data. DX is a commercial package produced and sold by IBM. It is not available at all Starlink sites. Your site manager should be able to advise on whether or not it is available at your site. If it is not available locally and you decide that you want to obtain a copy, Appendix B outlines the procedure.

This manual describes how to access DX and the Starlink enhancements to it. It also documents these enhancements. The Starlink enhancements to DX (collectively known as 'SX') have a number of purposes:

- they fill a few minor omissions in the functionality of basic DX,
- they provide easy-to-use functions to accomplish common tasks in a single step,
- they allow standard Starlink NDF data structures and data in other common astronomical formats to be imported into DX.

This edition of the manual applies to version 3.1 of DX and describes version 1.1 of SX. Normally SX will be installed at Starlink sites which have bought DX and this is assumed in the rest of this document.

2 MPEG Animations

SX contains a module (`SXMakeMPEG`) which allows individual images displayed by DX to be encoded into an MPEG animation. This module is based on the Berkeley `mpeg_encode` program, which must be available if the `SXMakeMPEG` module is to be used. If it is not currently installed on your machine, it can be obtained from various ftp sites. The latest version will be available at `mm-ftp.cs.berkeley.edu` in directory `/pub/multimedia/mpeg/encode`. However, obtaining software from ftp sites in the USA is usually a very slow business. For this reason, pre-built versions of `mpeg_encode` have been created and are available from ftp site `ast.man.ac.uk` in directory `outgoing/dsb`. To install a copy, retrieve the file appropriate to your operating system (`mpeg_encode-osf.tar.Z` or `mpeg_encode-sunos.tar.Z`) and decompress it using `uncompress`. The tar file contains the executable image and man page. These items should be extracted and copied to some suitable locations such as `/usr/local/bin` and `/usr/local/man/man1` respectively (your site manager will probably have to do this). Also available at `ast.man.ac.uk` is the postscript file `mpeg_encode.ps` which contains full documentation for `mpeg_encode`.

3 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[1].

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 (SX) should be automatically available when you start DX, and you can verify this by clicking on 'Import and Export' in the panel at the top left of the visual programming editor window, and then looking for some modules with names beginning with 'SX' in the panel at the bottom left of the visual programming editor window. If SX is not available then check that your site manager has installed it.

3.1 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. To re-instate the Starlink extensions, type the following command before starting DX:

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

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

4 Learning to Use DX

DX is a powerful and flexible tool 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. 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[2] 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'. The Starlink enhancements to DX include a number of networks for performing commonly required visualisations (see Sections 10 and 12) and make these tasks easier to perform. Also a variety of documentation is available; it is summarised in Table 1.

IBM Manuals

QuickStart Guide[1] (a set of introductory tutorials)
User's Guide[2]
User's Reference[3]
Programmer's Reference[4]

Starlink Documentation

SUN/203 *DX — IBM Data Explorer for Data Visualisation* (this document)
SC/2 *The DX Cookbook*[5]
SG/8 *An Introduction to Visualisation Software for Astronomy*[6]

On-line sources of information

A variety of on-line sources of information about DX are available; see SG/8 for a list.

Table 1: Documentation and sources of information for DX.

One possible route for learning to use DX is as follows.

1. Part I of SG/8 *An Introduction to Visualisation Software for Astronomy*[6] gives an overview of visualisation techniques, not tied to any particular package. If you have no prior knowledge of visualisation it may suggest techniques suitable for use with your data (DX supports most of the techniques described). Conversely, if you are familiar with

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

visualisation and know the techniques that you want to apply then you may omit this step.

2. SC/2 *The Starlink Data Explorer Cookbook*[5] provides an introduction to DX and recipes for common tasks. You should read the introduction and try any recipes which seem similar to your requirements. This document may well provide all the information that you require.

3. You can try the on-line tutorial available at the Department of Oceanography, University of Dalhousie (see SG/8[6]). However, the speed of the transatlantic computer networks is such that the tutorial will often be un-usably slow (even on a Sunday afternoon).

4. If none of the recipes in SC/2[5] are suitable then try the IBM *QuickStart Guide*[1], again following examples which seem similar to what you want to do. See the first entry in Section 5 below for a hint on running the examples in this guide.

5. If necessary, you can consult the IBM *User's Guide*[2] and *User's Reference*[3] for detailed information.

6. Finally, if you are experiencing difficulty using DX (and you are a Starlink user in the United Kingdom) then some advice and assistance is available from the Starlink Project. In the first instance contact David Berry (e-mail: `dsb@ast.man.ac.uk`). If he is unavailable then Clive Davenhall (e-mail: `acd@roe.ac.uk`) will act as a locum.

   An alternative source of expertise is the DX usenet news group:

   `comp.graphics.data-explorer`.

5 Useful Tips

This section lists various hints and tips which you may find useful when using DX. Additional tips are available in the `README` files supplied with DX. In a standard installation these files are in directory `/usr/lpp/dx`. In a non-standard installation they will still be in the `dx` directory, but this directory will have a different location in the host file system. The files available are:

`README` contains system independent information,

`README_`*xxx* contains system dependent information (*xxx* denotes a particular operating system, for example, `alphax` for Digital Unix/alpha).

1. There are a couple of cases where you may need to edit the tutorial files associated with the IBM *QuickStart Guide*[1] before they will run. In a standard installation the tutorial files and associated data files are located in the following directories:

   tutorials: `/usr/lpp/dx/samples/tutorial`
   data files: `/usr/lpp/dx/samples/data`

   In a non-standard installation DX will have the same directory tree beneath subdirectory `dx`, but will have a different location in the host file system.

- The tutorial files assume that DX has been installed in the recommended location (`/usr/lpp`, as above). If your system manager has installed it somewhere else you will need to edit the tutorial files so that they access data files in the correct location.

- A couple of tutorials refer to data file:

  `.../dx/samples/data/temp_wind.header`

  which does not exist. These tutorials should be edited to refer to file:

  `.../dx/samples/data/temp_wind.general`

2. The caps-lock key seems to alter the interpretation of keys which are not normally subject to the caps-lock setting. For example, you cannot delete a module icon from a network using <`control-delete`> when the caps-lock key is on. *When using DX the caps-lock key should always be off.*

3. When using DX on a Digital alpha, you may get an error message if you try to use the 'Image' or 'Display' modules, saying that 'gethostname' has failed. This problem only seems to occur if your display (either workstation or X-terminal) has no alias in the `/etc/hosts` file (or the equivalent NIS file). Just having an entry giving the numerical IP address of the display is insufficient; it must also have an alias.

4. Sometimes when DX is executed on a Sun running Solaris it crashes with the error message:

   ```
   cannot get shared memory segment
   ```

   There are two possible causes of this problem.

   - Under Solaris, DX uses shared memory in 256Mb segments. The default configuration for Solaris only allows 1Mb shared memory segments. To allow DX to run, your site manager should log in as `root`, and proceed as follows:

     (a) copy file `/etc/system` to `/etc/system.pre-dx` (that is, as a precaution, take a copy of the `system` file before editing it),

     (b) edit file `/etc/system` and add the line:

     ```
     set shmsys:shminfo_shmmax = 0x10000000
     ```

     (c) reboot the system.

- There is insufficient swap space available on the workstation[3]. You must either increase swap space or run DX with the `-memory` option to reduce its use of memory. For example, type `dx -memory 32` to restrict DX to 32Mb of memory.

5. Sometimes DX generates the error message:

   ```
   Out of memory
   ```

   The solution is to explicitly specify the (large) maximum amount of memory to be available for use when starting DX:

   `dx -memory` x

   where x is the amount of memory to be used, in Mb. Thus, for example, to specify a maximum of 200Mb you would type:

   ```
   dx -memory 200
   ```

   The most suitable value depends on the capabilities and configuration of your system. The following (possibly contradictory) criteria apply:

   - try to allocate at least two or three times the total size of your data set,
   - but do not allocate more than the total amount of swap space on your machine[3],
   - finally (but not least), other users will appreciate your setting a small value!

   Appendix A, *Using Data Explorer: Some Useful Hints*, in the IBM *User's Guide*[2] includes a section, *Memory Use*, which contains useful hints on reducing the amount of memory needed by DX.

6. If the 'Image' module runs without error but seems to produce no image (or if the image is too large or too small for the window), try resetting the image window by clicking anywhere inside it and then pressing <`control-f`>. The displayed object should be re-centred and re-sized within the window.

   If the 'Image' module repeatedly fails, generating X errors each time, then it may be necessary to log out and re-start the X window session.

[3]There are various Unix commands for showing the amount of swap space, but unfortunately they vary between Digital Unix/alpha and Solaris/Sun systems. Some of the possibilities are summarised in the following table:

Command	Digital Unix/alpha	Solaris/Sun
`swap -s`		•
`swap -l`		•
`swapon -s`	•	
`vmstat`	•	•

A bullet ('•') indicates that the command may be available on a system of the given type. Of course, whether or not it is actually available on your system depends on how the system and your personal search path have been configured. If the command is not immediately available in your search path then try prefixing the command with the directory specification '`/usr/sbin`'.

Unfortunately the command which is available for both types of system, `vmstat`, is the least useful because it does not directly show the amount of swap space available. Rather, it shows the number of free and used pages. Consult the appropriate `man` pages for further details.

7. If you are running DX from an X-terminal it will sometimes crash with X-windows errors if you attempt to run a network with two image windows. The solution is to run such networks from a workstation console.

8. If DX aborts for any reason (for example, you have typed <`control-c`>, the computer crashing etc.) it sometimes leaves rogue processes running. These processes will run indefinitely and consume large amounts of CPU time. You should check for them and then explicitly kill them. For example, on a Digital alpha you would check for them by typing something like:

```
ps aux | grep dx
```

9. Sometimes if your keyboard is not configured properly DX will generate a series of messages on start up as follows:

```
Could not convert X KEYSYM to a keycode
```

   These messages can be averted by using `xmodmap` to define a key for the required KEYSYM. For example, on a Digital alpha the following line could be edited into your `.X11Startup` file:

```
xmodmap -e "keycode 80 = End"
```

   which causes the F14 button on an N-108LK keyboard to generate the 'End' KEYSYM. The tool `xev` is useful for finding the key code associated with a given key on any keyboard.

10. If <`control-delete`> does not delete modules from a network (even when caps-lock is off), then create a file called `.motifbind` in your home directory. It should contain the following line.

```
osfDelete                 :<Key>Delete
```

   The problem should not re-occur after you have logged out and logged in again. Alternatively, to make the fix take effect immediately type:

```
% xmbind .motifbind
```

11. If DX crashes with an error message something like:

```
sendsig:  can't grow stack
```

   a likely cause is that you were attempting to execute a network with cyclic connections (that is, modules which directly or indirectly depend on their own output). If you try to connect modules in this way DX will usually issue a warning and refuse to generate the connection. However, it sometimes fails to detect and trap more complex and indirect instances of cyclic connection.

6 Importing Data in Astronomical Formats

There are two routes by which data in astronomical formats, such as the Starlink Extensible N-Dimensional Data Format (NDF; see SUN/33[7]), can be imported into DX. Either the data can be converted to DX format prior to starting DX or alternatively they can be accessed directly in their astronomical format from inside DX using some of the Starlink enhancements to DX. The former route is preferable if you plan to access the data more than once from within DX (which will usually be the case) because the conversion will be done just once; if you access the data in their original astronomical format from within DX the conversion is done 'on-the-fly' each time the data are read. Also, in practice, the first approach is perhaps simpler.

6.1 Conversion prior to using DX

The Starlink enhancements to DX include the application `ndf2dx` (see Appendix A) for converting a standard Starlink NDF data file into a format which can be accessed by DX. The output file generated is in the native DX format and it can be imported directly into DX.

There are various options available for `ndf2dx` but usually a Starlink NDF can be converted by simply typing:

`$SX_DIR/ndf2dx` *ndf_file* *dx_file*

where *ndf_file* is the name of the input NDF file to be converted and *dx_file* is the native DX format file created. Note that the input file is not altered in any way by this process. Also, by convention, the names of native DX format data files should end with the suffix '`.dx`'. The converted dataset may subsequently be accessed from within a DX visual program using the standard module 'Import'.

The usual, or 'native', method of storing an NDF dataset is as a file formatted using the Starlink Hierarchical Data System (HDS). However, in common with other Starlink applications, `ndf2dx` can read files in various other astronomical formats, such as FITS images, and interpret them as NDF datasets. By default `ndf2dx` will only read HDS files. In order to configure it to recognise files in other formats simply type:

```
convert
```

prior to using `ndf2dx`. Then simply run `ndf2dx` in the usual way, supplying the name of the input file, be it a FITS image, Figaro DST file or whatever, and any necessary conversions proceed automatically and invisibly. Table 2 lists some of the more common formats available. The set of data formats which is recognised is configurable. The default configuration and how to specify the configuration are both described in SUN/55[8].

6.2 Conversion within DX

An NDF data file can be accessed directly from within a DX visual program by including SX macro `SXReadNDF` (see Sections 10 and 11). The conversion is computed 'on-the-fly' as the data are read. Thus, if the data are to be accessed more than once (which will usually be the case) it is more efficient to perform the conversion prior to starting DX, as described in Section 6.1, above. If the conversion is done from within DX then the same set of alternative formats which are automatically available for prior conversions are also automatically available (see Table 2).

Data Format	File Type
'Native' NDF (Starlink HDS)	`.sdf`
FITS image	`.fit`
Figaro (version 2) DST	`.dst`
GASP	`.hdr`
IRAF image	`.imh`

Table 2: Valid data formats for input to `ndf2dx`. The file type serves to identify the format of a data file both to users and to `ndf2dx`.

7 Importing Data in Scientific Formats

DX contains facilities for importing data in the following scientific formats:

CDF – NASA Common Data Format,
NetCDF – NASA Network Common Data Format,
HDF – NCSA Hierarchical Data Format.

CDF and NetCDF are widely used in solar-terrestrial physics, but little used in other branches of astronomy. Importing data in these formats is documented in Appendix B, *Importing Data: File Formats*, in the IBM *User's Guide*[2].

8 Importing Data From Fortran Files

Before importing a data file into DX you must first create a general format header file which describes the structure of the data file. There are several ways of creating this header file. SC/2 *The Starlink Data Explorer Cookbook*[5] gives a couple of examples. Section 4 of the IBM *QuickStart Guide*[1] contains further examples; in particular all the keywords which can occur in the header file are described in Section 4.3. Note that the name of the data file is one of the items of information which must be included in the general format header.

8.1 Formatted files

Once a suitable general format header file has been created, a formatted file written by a Fortran program can be imported into DX using the standard module 'Import'.

8.2 Unformatted files

Unfortunately DX cannot directly read an unformatted file written by a Fortran program. Such files contain record control bytes which must be removed prior to importing the file into DX. However, the first stage of importing such a file is still to create a general format header file. The `format` keyword should be either '`ieee`' or '`binary`'.

Once the header file has been created there are two ways to proceed.

First method

1. Use `SXUnfort` to remove the record control bytes prior to starting DX. From the Unix command line type:

   `$SX_DIR/SXUnfort` *input_file* *output_file*

   input_file is a Fortran unformatted file and *output_file* is the converted file.

2. A general header file can then be created to describe the data. Note that the data file name entered into the general header file should be the name of the converted file (*output_file*, above).

3. Import the converted file (*output_file*) into DX using the standard module 'Import'.

Second method

Create a general header file, including the name of the raw Fortran data file, and then use the SX macro `SXReadFortran` (see Sections 10 and 11) to import the unformatted file directly into DX. The record control bytes are stripped 'on-the-fly' each time the data are read. Thus, if the data are to be accessed more than once (which will usually be the case) it is more efficient to perform the conversion prior to starting DX, as described in the first method, above.

The procedure for importing Fortran data files is identical on both Digital Unix/alpha and Solaris/Sun. 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 in the general header file[4]. See section 4.3 of the IBM *QuickStart Guide*[1] for details.

9 Exporting NDFs From DX

DX is a visualisation package whose purpose is to display and examine data. Usually you will use it to visualise the final set of data computed at the end of a sequence of reductions and analyses or theoretical calculations. Hence you are unlikely to want to convert a dataset in the native DX format into the Starlink NDF format for further processing. However, occasionally you may want to carry out such a conversion, and it is possible in some cases. The native DX format is powerful, flexible and general and it cannot be successfully converted to a Starlink NDF in all cases.

To convert a native DX format file into a Starlink NDF use the 'SXList' module (see Sections 10 and 11) to produce a text file containing the required data. Then use KAPPA application `trandat` (see SUN/95[9]) or CONVERT application `ascii2ndf` (see SUN/55[8]) to convert the text file to an NDF.

[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.

10 SX – The Starlink Enhancements to DX

This and subsequent sections assume that you have a basic familiarity with DX. If you are in doubt then see, for example, SC/2[5].

There are three types of Starlink enhancement to DX: modules, macros and demonstration networks. The modules and macros fill a few omissions in the functionality of basic DX. The demonstration networks are easy-to-use examples of how to perform common tasks.

As a user you will not notice much difference between modules and macros. Both perform some defined task and appear as an icon which may be positioned within the canvas of the DX visual programming editor. Technically the difference is that modules provide entirely new functionality and are written in C, whereas macros are scripts invoking several existing modules, though normally you will not be aware of these details. There are, however, some slight differences in the way modules and macros behave. For example, if you click on a macro and then select the 'Open Selected Macros' item from the 'Windows' menu it will expand into a diagram showing its constituent components. Also the way that on-line help is provided is slightly different. Because of these differences the type of each item (module or macro) is indicated in its detailed documentation.

The following lists give one-line summaries for the modules and macros and the demonstration networks. Subsequent sections document the individual items in detail.

10.1 Alphabetical list of modules and macros

The Starlink DX modules and macros are summarised alphabetically below. See Section 11 for a detailed description of each item.

SXBin – bins points into a supplied grid.

SXBounds – finds the corners of a field's bounding box.

SXConstruct – constructs a regular field with regular connections.

SXDummy – a dummy module which does nothing.

SXEnum – enumerates the positions or connections in a field.

SXList – write values from a field or list to a text file.

SXMakeMpeg – constructs an MPEG animation.

SXPercents – searches a histogram of data values for requested percentiles.

SXPrint – creates a postscript version of a displayed image.

SXProfile – creates a one-dimensional profile through a field between two points.

SXRand – creates a set of random vectors.

SXReadFortran – reads a Fortran binary data file.

SXReadNDF – reads data from a Starlink NDF structure.

SXRegrid – samples a field at positions defined by another field.

SXSubset – creates a rectangular or arbitrary subset of a field.

SXVisible – identifies visible positions.

10.2 Alphabetical list of demonstration networks

The Starlink DX demonstration networks are summarised alphabetically below. See Section 12 for a detailed description of each network.

iso – displays iso-surfaces taken from a regular data grid.

scatter – displays scattered particle data. The data may be mapped on to a regular grid and written to a disk file.

slice – displays two-dimensional slices through a three-dimensional regular data grid.

stream – displays a three-dimensional vector field defined on a regular grid as a set of stream lines.

11 Starlink DX Modules and Macros

This section presents a detailed description for each Starlink DX module and macro. The entries are listed alphabetically. The format in which the information for each module and macro is presented is deliberately similar to that used for the standard modules in the IBM *User's Reference*[3]. The examples for several of the modules and macros use example data files included in the standard release of DX. If your copy of DX has been installed in a non-standard location then see the first entry in Section 5 for details of how to find these files.

SXBin Realization

Name

SXBin – bins points into a supplied grid

Syntax

output = `SXBin`(**input, grid, type**);

Inputs

Name	Type	Default	Description
input	field or group	none	field or group with positions to bin
grid	field	none	grid to use as template
type	integer	0	type of output values required: 0 - mean data value in bin 1 - sum of data values in bin 2 - bin population

Outputs

Name	Type	Description
output	field or group	binned field

Description

The `SXBin` module bins the **data** component of the **input** field into the bins defined by the "connections" component of the **grid** field. The input field can hold scattered or regularly gridded points, but the "data" component must depend on "positions". The **grid** field must contain "connections" and "positions" components but need not contain a "data" component. The input "data" component must be either `TYPE_FLOAT` or `TYPE_DOUBLE`.

The "data" component in the **output** field contains either the mean or sum of the **input** data values falling within each connection, or the number of data values falling within each connection, as specified by **type**.

When binning a regular grid into another regular grid, beware of the tendency to produce artificial large scale structure representing the "beat frequency" of the two grids.

Components

All components except the "data" component are copied from the **grid** field. The output "data" component added by this module depends on "connections". An "invalid connections" component is added if any output data values could not be calculated (e.g. if the mean is required of an empty bin).

Examples

This example bins the scattered data described in file "CO2.general" onto a regular grid, and displays it. SXBin is used to find the mean data value in each grid connection:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
camera = AutoCamera(frame17);
grid = SXConstruct(frame17,deltas=10);
bin = SXBin(frame17,grid);
coloured = AutoColor(bin);
Display(coloured,camera);
```

The next example produces a grid containing an estimate of the density of the scattered points (i.e. the number of points per unit area). The positions of the original scattered points are shown as dim grey circles. SXBin finds the number of input positions in each bin, Measure finds the area of each bin, and Compute divides the counts by the areas to get the densities:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
camera = AutoCamera(frame17);
glyphs=AutoGlyph(frame17,scale=0.1,ratio=1);
glyphs=Color(glyphs,"dim grey");
grid = Construct([-100,-170],deltas=[40,40],counts=[6,10]);
counts = SXBin(frame17,grid,type=2);
areas = Measure(counts,"element");
density = Compute("$0/$1",counts,areas);
coloured = AutoColor(density);
collected=Collect(coloured,glyphs);
Display(collected,camera);
```

See Also

SXRegrid, Map, Construct, SXConstruct, Measure

SXBounds (*macro*) **Structuring**

Name

SXBounds – Finds the corners of a field's bounding box

Syntax

lower, upper = `SXBounds`(**input**);

Inputs

Name	Type	Default	Description
input	field	none	input field

Outputs

Name	Type	Description
lower	vector	lower bounds
upper	vector	upper bounds

Description

The `SXBounds` macro returns the upper and lower bounds of the box enclosing the supplied **field.** The bounds are obtained from the "box" component of the supplied field.

SXConstruct Realization

Name

SXConstruct – constructs a regular field with regular connections

Syntax

output = `SXConstruct`(**object, lower, upper, deltas, counts**);

Inputs

Name	Type	Default	Description
object	field	no default	object to define extent of new field
lower	vector	no default	explicit lower bounds of new field
upper	vector	no default	explicit upper bounds of new field
deltas	scalar or vector	no default	increment for each axis
counts	integer or vector	no default	number of positions along each axis

Outputs

Name	Type	Description
output	field	output field

Description

The `SXConstruct` module constructs a field with regular positions and connections covering a volume with specified bounds. It is similar to the standard `Construct` module, but is somewhat easier to use if a simple grid is required.

If **object** is given, its bounds define the extent of the output field. Otherwise, the vectors given for **upper** and **lower** define the extent of the output field.

If **deltas** is supplied, it defines the distances between adjacent positions on each axis. It should be a vector with the same number of dimensions as **upper** and **lower**, or a single value (in which case the supplied value is used for all axes). The upper and lower bounds are expanded if necessary until they span an integer number of deltas.

If **deltas** is not supplied, then **counts** must be supplied and should be an integer vector giving the number of positions on each axis, or a single integer (in which case the same value is used for all axes).

Components

The **output** has "positions", "connections" and "box" components, but no "data" component.

Examples

This example imports a scattered data set, extracts a single frame, uses `SXConstruct` to make a grid covering the bounds of the imported data set, with increments of 10.0 along each axis, and then uses `SXBin` to find the mean data value in each of the square connections of this new grid. The resulting field is displayed:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
newgrid = SXConstruct(frame17,deltas=10);
binned = SXBin(frame17,newgrid);
coloured = AutoColor(binned);
camera = AutoCamera(binned);
Display(coloured,camera);
```

See Also

`Construct`, `Grid`

SXDummy **Special**

Name

SXDummy – a dummy module which does nothing

Syntax

= `SXDummy`(**field, group, vector, scalar, integer, vectorlist, scalarlist, integerlist, string, camera**);

Inputs

Name	Type	Default	Description
field	field	none	an input field
group	group	none	an input group
vector	vector	none	an input scalar
scalar	scalar	none	an input scalar
integer	integer	none	an input integer
vectorlist	vector list	none	an input vector list
scalarlist	scalar list	none	an input scalar list
integerlist	integer list	none	an input integer list
string	string	none	an input string
camera	camera	none	an input camera

Description

The `SXDummy` module does nothing. It is used merely to force inputs to macros to be of a specified type. Connecting the output from an `Input` module to one of the inputs of `SXDummy` forces the `Input` to be of the same type as the `SXDummy` input to which it is connected.

SXEnum Realization

Name

SXEnum – enumerates the positions or connections in a field

Syntax

output = `SXEnum`(**input, name, dep**);

Inputs

Name	Type	Default	Description
input	field or group	none	field or group to enumerate
name	string	"`data`"	name of component to store the enumeration
dep	string	"`positions`"	object to be enumerated; "`positions`" or "`connections`"

Outputs

Name	Type	Description
output	field or group	enumerated field

Description

The `SXEnum` module creates the component specified by **name,** and adds it to the **output** field. The values in the new component start at zero and increment by one for each position or connection in the field.

The new component is in one-to-one correspondence with either the "positions" or "connections" component, dependent on **dep.**

Components

Adds a component with the given **name,** deleting any existing component. All other components are copied from the **input** field.

Examples

In this example, the 17th frame is extracted from the scattered data set described in file "`CO2.general`", and a field created holding only those positions with offsets between 10 and 20:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
enum = SXEnum(frame17,"index","positions");
marked = Mark(enum,"index");
included = Include(marked,10,20,1);
subset = Unmark(included,"index");
```

SXList — Import and Export

Name

SXList – write values from a field or list to a text file

Syntax

nrec = `SXList`(**input, file, append, names**);

Inputs

Name	Type	Default	Description
input	field or value or value list or string or string list	none	input field or list
file	string	(Message Window)	name of the text file
append	integer	0	1 to append to existing file 0 to create new file
names	string or string list	"`data`"	the field components to list

Outputs

Name	Type	Description
nrec	integer	number of records written to the file

Description

The `SXList` module writes numerical or string values to an output text **file.** The values may be obtained from a field, or they may be specified explicitly as a list. Each record written to the **file** contains one item from the list, or one item from each of the field components specified by **names,** layed out in columns.

It is not an error to give a null **input,** but nothing will be written to the file.

If no value is supplied for **file** then the list will be displayed in the message window.

If **append** is set to 1, the output will be appended to any existing **file** with the given name. A new file will be created if none exists. If **append** is set to 0, a new file is created every time, potentially over-writing an existing file.

Examples

In this example, the "data" and "positions" components from a field are written out to a text file called "`dump.dat`". The file starts with a header containing three comment lines:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
```

```
SXList("#","# Positions and data","#","dump.dat",0);
SXList(frame17,"dump.dat",1,"positions","data");
```

See Also

Export, Echo, Print

SXMakeMpeg (*macro*) Import and Export

Name

SXMakeMpeg – constructs an MPEG animation

Syntax

= `SXMakeMpeg`(**image, save, mpeg**);

Inputs

Name	Type	Default	Description
image	image	none	image holding the next frame of the animation
save	flag	0	0 to add another frame, 1 to save the frames to an MPEG
mpeg	string	none	name of an MPEG file, including a file suffix (e.g. ".mpg")

Description

Each time the `SXMakeMpeg` macro is activated with **save** set to 0, the supplied **image** is converted into a "`PPM`" file on disk in the current directory, with a name like `SX_frame.n.ppm` (where `n` is a frame number). When `SXMakeMpeg` is called with **save** set to 1, these `PPM` files are encoded into an MPEG animation with name given by **mpeg**, using the public domain Berkeley encoder "`mpeg_encode`", which should be installed on your host system (it can be obtained from ftp site `mm-ftp.cs.berkeley.edu` in directory `/pub/multimedia/mpeg/encode`, or from ast.man.ac.uk in directory outgoing/dsb). The `PPM` files and other intermediate files such as the file holding the encoder parameter values, are deleted once the MPEG has been created.

The input **image** must be a rendered image, as created by the `Render` module.

Giving **save** as 1, without supplying a value for **mpeg** causes the current `PPM` frames to be deleted without attempting to create an MPEG.

While `mpeg_encode` is running, it displays statistics and progress messages in the DX message window. Note, it can take a long time to encode an MPEG containing many frames, especially if each frame is large. It is best to use small frames, partly to speed up the encoding, but also because the resulting animation will play back faster, and occupy less disk space.

Note, only one instance of the `SXMakeMpeg` module can be included in a DX network.

Examples

This example creates an MPEG from the first 3 frames of the data described in file "`CO2.general`". The image size is reduced to 320 pixels by setting the **resolution** in `AutoCamera` in order to speed up the display of the movie.

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
coloured = AutoColor(input);
cam = AutoCamera(input,resolution=320);

frame = Select(coloured,1);
image = Render(frame,cam);
SXMakeMpeg(image);

frame = Select(coloured,2);
image = Render(frame,cam);
SXMakeMpeg(image);

frame = Select(coloured,3);
image = Render(frame,cam);
SXMakeMpeg(image);

SXMakeMpeg(save=1,mpeg="CO2.mpg");
```

See Also

WriteImage

SXPercents Transformation

Name

SXPercents – searches a histogram of data values for requested percentiles

Syntax

outputlist, output = `SXPercents`(**input, percents**);

Inputs

Name	Type	Default	Description
input	field	none	field containing the histogram
percents	scalar list	{5.0,95.0}	the required percentiles

Outputs

Name	Type	Description
outputlist	scalar list	list containing the data values at all the requested percentiles
output	scalar	data value at a single requested percentile
...		more single data values

Description

The `SXPercents` module searches the histogram supplied by **input,** for the data values corresponding to the requested percentiles. The **input** histogram should have been created using the `Histogram` module.

Each value supplied in **percents** should be a value in the range 0.0 to 100.0. For each value, P, the corresponding **output** value is the data value below which P percent of the data values in the histogram lie. Linear interpolation is performed to find this value.

The **outputlist** parameter is a list containing all the individual **output** values.

Examples

This example displays the electron density field, with a colour table which ignores the lowest and highest 2 percent of the data values:

```
electrondensity = Import("/usr/lpp/dx/samples/data/watermolecule");
histogram = Histogram(electrondensity)
plist, lo, hi = SXPercents(histogram,2.0,98.0);
camera = AutoCamera(electrondensity);
coloured = AutoColor(electrondensity,min=lo,max=hi);
Display(coloured,camera);
```

See Also

`Histogram`, `Statistics`

SXPrint (*macro*) Import and Export

Name

SXPrint – creates a postscript version of a displayed image

Syntax

outlen = `SXPrint`(**object, camera, file, width, dpi, length, encapsulated, colour, portrait**);

Inputs

Name	Type	Default	Description
object	object	none	object to be rendered
camera	camera	none	camera used to display **object**
file	string	none	output file name
width	scalar	8	width of printed image in inches
dpi	integer	300	dots per inch
length	value list	1.0	a length at old resolution
encapsulated	integer	0	produce an encapsulated file? (0=no, 1=yes)
colour	integer	1	produce a colour file? (0=no, 1=yes)
portrait	integer	0	use portrait mode? (0=no, 1=yes)

Outputs

Name	Type	Description
outlen	value list	**length** at new resolution

Description

The `SXPrint` macro renders the supplied object using the supplied camera and creates a postscript file holding the image. The postscript version is rendered at the resolution specified by **dpi** to produce a picture with the supplied **width** (in inches).

The **object** and **camera** will usually be obtained from the output parameters of the `Image` module.

Note, the size and shape of some objects are specified in terms of pixels (e.g. the objects created by `Colorbar` and `Caption`) and these objects will *not* be re-rendered to the higher resolution. They will thus appear smaller in the printed version of the image than on the screen. To get round this problem, module `SXPrint` can be used to calculate new sizes for such objects which will cause them to retain their original relative sizes in the printed version. To do this, assign the original object sizes (as specified in the configuration box of the module which produces the object) to **length**, and do not assign any values to **file** or **object** (all other parameters should be set to the values they will have when the final postscript file is produced). The **outlen** parameter will then be returned

holding the equivalent object sizes at the higher resolution. These should then be passed on to `Colorbar`, `Caption`, etc, to produce objects of the correct size.

See Also

`Image`, `WriteImage`

SXProfile (*macro*) Realization

Name

SXProfile – creates a 1-dimensional profile through a field between 2 points

Syntax

line, profile = `SXProfile`(**input, start, end, density**);

Inputs

Name	Type	Default	Description
input	field	none	input data field
start	vector	lower bounds of input field	starting point of line
end	vector	upper bounds of input field	ending point of line
density	integer	30	number of points along line

Outputs

Name	Type	Description
line	geometry field	geometry field describing the line
profile	field	1D field holding data values along the line

Description

The `SXProfile` macro samples the **input** field at regular intervals along a straight line joining the **start** and **end** positions. The number of samples is given by **density**.

The sampled data values are returned in **profile**, which has a "positions" component holding the distance from the **start** to each sample. This field can be converted to a 1-D plot using the `Plot` module, which can then be viewed using the `Image` module. To obtain a hard-copy, use the `Print Image` option in the `File` menu of the `Image` window. Superior plots can be obtained by dumping the profile to an ASCII file using `SXList`, and then using a plotting program such as `gnuplot`, `sm` or `Mongo` to produce the plot. See SG/8 for an overview of available plotting packages.

Two vector-valued attributes named "profile_start" and "profile_end" are added to the output **profile**, holding the **start** and **end** vectors.

A geometry field describing the sampled positions in the original n-D space is returned in **line**. This could, for instance, be included with a rendering of the original data to indicate the position of the profile.

Examples

This example displays a profile through the `watermolecule` field containing 50 values sampled between positions $[-1.0, -1.0, -1.0]$ and $[1.0, 1.0, 0.0]$. The profile is converted into a plot using `Plot`. A title caption is created and collected together with the plot, before displaying with `Display`.

```
electrondensity = Import("/usr/lpp/dx/samples/data/watermolecule");
line, profile = SXProfile(electrondensity,[-1.0,-1.0,-1.0],[1.0,1.0,0.0],50);
plot = Plot(profile,"Distance along profile","Electron density",
            labelscale=0.5,aspect=0.75,frame=2);
camera = AutoCamera(plot,width=3.5);
caption = Caption("Profile from [-1.0,-1.0,-1.0] to [1.0,1.0,0.0]",
            [0.7,0.99],height=18);
collected = Collect(plot,caption);
Display(collected,camera);
```

See Also

Plot, Image, SXList

SXRand Realisation

Name

SXRand – create a set of random vectors

Syntax

output = `SXRand`(**nvec, ndim, dist, a, b**);

Inputs

Name	Type	Default	Description
nvec	integer or field or group	1	number of vectors required
ndim	integer	1	dimensionality for each vector
dist	integer	1	distribution: 1 - uniform 2 - normal 3 - Poisson
a	scalar	0.0	first parameter describing the distribution
b	scalar	1.0	second parameter describing the distribution

Outputs

Name	Type	Description
output	vector list or field	output vectors

Description

The `SXRand` module creates a set of random vectors. Each vector has **ndim** components, each chosen independently from the distribution specified by **dist.** The vectors can be returned as a list of **nvec** vectors (if **nvec** is an integer), or as the "data" component of a field (if **nvec** is a field).

If **dist** is 1, then a uniform distribution between **a** and **b** is used. If **dist** is 2 then a normal distribution with mean **a** and standard deviation **b** is used. If **dist** is 3, then a Poisson distribution is used with mean **a.** Integer values are returned for a Poisson distribution, and double precision values are returned for uniform and normal distributions.

Components

If **nvec** is a field, a "data" component is added to the output field, replacing any existing "data" component. The output data will be dependent on "positions" if the input field had no "data" component. Otherwise, it will have the same dependency as the input field. Any components which are dependent on "data" are modified. All other components are copied from the input field.

Examples

This example constructs a regular grid of 50 by 50 by 50 points containing 3-vectors at each position in which each component is uniformly distributed between 0 and 1:

```
newfield = Construct([0,0,0],[1,1,1],[50,50,50]);
rand = SXRand( newfield, 3 );
```

This example generates and displays a list of 5 samples from a normal distribution of mean 10 and standard deviation 2:

```
randvals = SXRand( 5, 1, 2, 10.0, 2.0 );
Echo( randvals );
```

SXReadFortran (*macro*) Import and Export

Name

SXReadFortran – reads a Fortran binary data file

Syntax

data = `SXReadFortran`(**header, variable, start, end, delta**);

Inputs

Name	Type	Default	Description
header	string	none	general format header file
variable	string or string list	everything	variables to be read
start	integer	first frame	starting data frame
end	integer	last frame	ending data frame
delta	integer	1	increment between frames

Outputs

Name	Type	Description
data	object	object containing requested variables

Description

The `SXReadFortran` macro imports binary data files written by Fortran unformatted `WRITE` statements. It strips the Fortran record control bytes out of the binary data before importing it. *NB*, Fortran formatted files (i.e. text files) should be imported using the `Import` module.

The supplied value for **header** should be a text file containing a general format header describing the binary data. The actual data should be in a separate file identified by the usual "file" keyword in the header.

The conversion is done "on-the-fly" each time a binary file is accessed. As an alternatively to using the `SXReadFortran` macro, the `SXUnfort` program can be used to perform the conversion once, after which the `import` module can be used to import the converted file. See the prologue of `$SX_DIR:SXUNfort.c` for more details.

The other parameters are equivalent to the corresponding parameters of the `Import` module (the **format** parameter of the `Import` module is fixed at "general").

See Also

`Import`

SXReadNDF (*macro*) Import and Export

Name

SXReadNDF – reads data from a Starlink NDF structure

Syntax

data = `SXReadNDF`(**ndf, positions**);

Inputs

Name	Type	Default	Description
ndf	string	none	the input NDF structure
positions	integer	1	data dependency: 0 - connections 1 - positions

Outputs

Name	Type	Description
data	object	data read from **ndf**

Description

The `SXReadNDF` macro imports data from a Starlink NDF structure. It uses the `$SX_DIR/ndf2dx` application.

The dependency of the data values in the NDF is specified by the **positions** parameter.

The conversion is done "on-the-fly" each time an NDF is accessed. As an alternatively to using the `SXReadNDF` macro, the `ndf2dx` application can be used to perform the conversion once, after which the `import` module can be used to import the converted file.

The facilities provided by the NDF library for transparently accessing foreign data formats are available with this module. See Section 5.

See Also

`Import`

SXRegrid Realization

Name

SXRegrid – samples a field at positions defined by a another field

Syntax

output = `SXRegrid`(**input, grid, nearest, radius, scale, exponent, coexp, type**);

Inputs

Name	Type	Default	Description
input	field or group	none	field or group with positions to regrid
grid	field	none	grid to use as template
nearest	integer or string	1	number of nearest neighbours to use, or "`infinity`"
radius	scalar or string	"`infinity`"	radius from grid point to consider, or "`infinity`"
scale	scalar or vector or scalar list	1.0	scale lengths for weights
exponent	scalar	1.0	weighting exponent
coexp	scalar	0.0	exponential co-efficient for weights
type	integer	0	type of output values required: 0 - weighted mean 1 - weighted sum 2 - sum of weights

Outputs

Name	Type	Description
output	field or group	regridded field

Description

The `SXRegrid` module samples the "data" component of the **input** field at the positions held in the "positions" component of the **grid** field. It is similar to the standard `Regrid` module, but provides more versatility in assigning weights to each input position, the option of returning the sums of the weights or the weighted sum instead of the weighted mean, and seems to be much faster. Both supplied fields can hold scattered or regularly gridded points, and need not contain "connections" components. The "data" component in the **input** field must depend on "positions".

For each grid position, a set of near-by positions in the input field are found (using **nearest** and **radius**). Each of these input positions is given a weight dependent on its distance from the current grid position. The output data value (defined at the grid position) can be the weighted mean or weighted sum of these input data values, or the sum of the weights (selected by **type**).

The weight for each input position is of the form $(d/d_0)^p$ where d is the distance from the current grid position to the current input position, and p is the value of **exponent.** If a single value is given for **scale** then that value is used for the d_0 constant for all the near-by input positions. If more than 1 value is given for **scale** then the first value is used for the closest input position, the second value for the next closest, etc. The last supplied value is used for any remaining input positions. A value of zero for **scale** causes the corresponding input position to be given zero weight.

If **coexp** is not zero, then the above weights are modified to become $\exp((d/d_0)^p)$.

If **nearest** is given an integer value, it specifies N, the maximum number of near-by input positions to use for each output position. The N input positions which are closest to the output position are used. If the string "`infinity`" is given, then all input positions closer than the distance given by **radius** are used. Using **radius,** you may specify a maximum radius (from the output position) within which to find the near-by input positions. If the string "`infinity`" is given for **radius** then no limit is placed on the radius.

Components

All components except the "data" component are copied from the **grid** field. The output "data" component added by this module depends on "positions". An "invalid positions" component is added if any output data values could not be calculated (e.g. if there are no near-by input data values to define the weighted mean, or if the weights are too large to be represented, or if the input grid position was invalid).

Examples

This example maps the scattered data described in file "`CO2.general`" onto a regular grid and displays it. `SXRegrid` is used to find the data value at the nearest input position to each grid position:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
camera = AutoCamera(frame17);
grid = Construct([-100,-170],deltas=[10,10],counts=[19,34]);
regrid = SXRegrid(frame17,grid);
coloured = AutoColor(regrid);
Display(coloured,camera);
```

The next example produces a grid containing an estimate of the density of the scattered points (i.e. the number of points per unit area). The positions of the original scattered points are shown as dim grey circles. `SXRegrid` finds the 5 closest input positions at each grid position. Zero weight is given to the closest 3 positions. The fourth position has a weight which is half the density of the points within the circle passing through the fourth point (i.e. if the fourth point is at a distance D from the current grid position, there are 3 points within a circle of radius D, so the density within that circle is $3/(\pi.D^2)$). The fifth position has a weight which is half the density of the points within the circle passing through the fifth point. The output data value is the sum of the weights (because **type** is set to 2), which is the mean of the densities within the circles touching the fourth and fifth points:

```
input = Import("/usr/lpp/dx/samples/data/CO2.general");
frame17 = Select(input,17);
camera = AutoCamera(frame17);
glyphs = AutoGlyph(frame17,scale=0.1,ratio=1);
glyphs = Color(glyphs,"dim grey");
grid = Construct([-100,-170],deltas=[10,10],counts=[19,34]);
density = SXRegrid(frame17,grid,nearest=5,scale=[0,0,0,0.691,0.798],
exponent=-2,type=2);
coloured = AutoColor(density);
collected = Collect(coloured,glyphs);
Display(collected,camera);
```

See Also

SXBin, ReGrid, Map, Construct, SXConstruct

SXSubset (*macro*) Import and Export

Name

SXSubset – creates a rectangular or arbitrary subset of a field

Syntax

output = `SXSubset`(**input, object, lower, upper, expression**);

Inputs

Name	Type	Default	Description
input	field	none	the field to be subsetted
object	field	object assigned to **input**	object to define corners of rectangular subset
lower	vector	lower bounds of **object**	lower bounds of rectangular subset
upper	vector	upper bounds of **object**	upper bounds of rectangular subset
expression	string	none	`Compute` expression specifying an arbitrary subset

Outputs

Name	Type	Description
output	field	the subset

Description

The `SXSubset` macro returns a subset of the **input** field, containing either the positions which fall within the rectangular volume specified by **object, lower** and **upper,** or the positions for which the given **expression** evaluates to a non-zero answer. The **input** field can have either scattered or gridded positions, and need not have a "connections" component.

If a value is supplied for **expression** then **object, lower** and **upper** are ignored. In this case, **expression** should be a string holding an arithmetic or logical expression suitable for use by the `Compute` module. It can contain references to the variables "field" (`$0` in the scripting language) and "index" (`$1` in the scripting language) . The "field" variable refers to the "positions" component of the **input** field, and so "field.x", "field.y" and "field.z" can be used to refer to the (x, y, z) values at each **input** position. The "index" variable is a scalar with integer values and refers to an enumeration of the **input** "positions" component (i.e. "index" is 0 at the first position, 1 at the second, 2 at the second, etc). *N.B.*, **expression** should not refer to components of "field" which do not exist (i.e. do not refer to "field.z" if **input** is 2-dimensional).

If no **expression** is supplied, **lower** and **upper** are used to define a rectangular brick with the given bounds. **lower** and **upper** default to the bounds of **object.** *Note*, **lower** and **upper** should have the correct number of components (i.e. 2 for a 2-dimensional **input,** 3 for a 3-dimensional **input**)

Components

Modifies the "data", "positions" and "connections" and any components which depend on "positions" or "connections". All other components are propagated to the output.

Examples

This example displays the `watermolecule` field between bounds of $[-1.0, -1.0, -1.0]$ and $[1.0, 1.0, 0.0]$. The `ClipBox` module could have been used to do this, but `ClipBox` does not actually remove the excluded positions from the data set; it just tells the renderer not to render them:

```
electrondensity = Import("/usr/lpp/dx/samples/data/watermolecule");
subset = SXSubset(electrondensity,lower=[-1.0,-1.0,-1.0],upper=[1.0,1.0,0.0]);
volume = AutoColor(subset,min=0,max=0.5);
camera = AutoCamera(volume,"off-diagonal");
Display(volume,camera);
```

The next example creates a translucent isosurface of the `watermolecule` field, and then takes a subset of the field which matches the bounds of the isosurface. The coloured volume and the isosurface are collected together and displayed:

```
electrondensity = Import("/usr/lpp/dx/samples/data/watermolecule");
isosurface = Isosurface(electrondensity,0.3);
isosurface = Color(isosurface,"grey",0.3);
subset = SXSubset(electrondensity,isosurface);
volume = AutoColor(subset,min=0,max=0.5);
both = Collect(volume,isosurface);
camera = AutoCamera(both,"off-diagonal");
Display(both,camera);
```

The next example, creates a spherical subset of the `watermolecule` field, centred on $[1.0, 0, -1]$, of radius 0.5, and displays it face on. In the `SXSubset` expression, the squared radius of each position is compared 0.25 (i.e. the square of 0.5):

```
electrondensity = Import("/usr/lpp/dx/samples/data/watermolecule");
subset = SXSubset(electrondensity,expression="$0.x**2+$0.y**2+$0.z**2<0.25");
volume = AutoColor(subset,min=0,max=0.5);
camera = AutoCamera(volume);
Display(volume,camera);
```

The next example, creates a subset of the scattered data described in file "`CO2.general`", containing every second element, starting at element zero. `SXSubset` uses the modulus function in `Compute` ("`%`") to assign a value of 1 to the required positions, and zero to the other positions:

```
data = Import("/usr/lpp/dx/samples/data/CO2.general");
subset = SXSubset(data,expression="($1+1)%2");
glyphs = AutoGlyph(subset);
camera = AutoCamera(glyphs);
Display(glyphs,camera);
```

See Also

`Compute`, `Slab`, `Slice`, `ClipBox`

SXVisible (*macro*) Transformation

Name

SXVisible – identifies visible positions

Syntax

outfield, flags = `SXVisible`(**field, list, camera**);

Inputs

Name	Type	Default	Description
field	field	none	field containing positions to be checked
list	vector list	none	list of positions to be checked
camera	camera	none	camera used to view positions

Outputs

Name	Type	Description
outfield	field	field with invisible positions removed
flags	byte list	visibility flags for each position

Description

The `SXVisible` macro determines which of the positions given by **field** or **list** would be visible if viewed with the projection defined by **camera**. A list of bytes is returned in **flags** which corresponds one-for-one with the input positions. This list contains 1 if the corresponding position is visible and 0 otherwise.

The input positions are defined by the "positions" component of **field** if **field** is provided, or by **list** otherwise. A copy of the supplied **field** (if any) is returned in **outfield** from which all invisible positions have been removed.

This module works with both perspective and orthographic cameras.

Components

All components of **field** are propagated to the **outfield**.

See Also

`AutoCamera`, `Image`

12 Starlink DX Demonstration Networks

Several demonstration networks are provided with SX which provide ready-to-go facilities for generating simple visualisations, and also provide a starting point for your own networks. The demonstrations available are:

iso – displays iso-surfaces taken from a regular data grid. The network is in file `$SX_DIR/iso_demo.net`.

slice – displays 2-d slices through a 3-d regular data grid. The network is in file `$SX_DIR/slice_demo.net`.

stream – displays a 3-d vector field defined on a regular grid as a set of stream lines. The network is in file `$SX_DIR/stream_demo.net`.

scatter – displays scattered particle data. The data may be mapped on to a regular grid and written to a disk file. The network is in file `$SX_DIR/scatter_demo.net`.

Sample data files are also available:

`/usr/lpp/dx/samples/data/storm_data.dx` – contains vector and scalar fields defined on a 3-d regular grid.

`/star/etc/sx/scattered_data.1.dx` – contains vector and scalar fields defined at scattered positions in 3-d space. If the Starlink software collection is not rooted at `/star`, then replace `/star` with the actual Starlink root directory. There are eight other scattered data files in the same directory, with names in which the "`1`" in the above file name is replaced by an integer between 2 and 9.

All these files are in native DX format.

12.1 Running the Demonstration Networks

The demonstration networks can be read into DX like any other network, using the "Open Program..." option in the "File" menu of the network editor window. Alternatively, they can be executed more simply by starting dx with the command:

```
% dx demo <name>
```

where `<name>` is the name of the demonstration to be run. If no name is given, the user is given a list of available demonstrations and prompted for a name before continuing. Using this access method, the demonstration is presented as a finished application, with help provided to guide the user through its use. The gory details of the network are hidden away, and all the user sees are the "front-end" controls used to control the behaviour of the network. Later sections of this document describe the controls available for each network.

12.1.1 Executing the network

Each demonstration is basically a large program with several input parameters, which runs from start to finish without intervention each time it is executed, using the parameter values established by the user before execution commenced[5]. To use a demonstration, all parameters should be set to the required values (or left at their default values), and the program should then be executed once by selecting the "Execute once" item from the "Execution" menu in the Image window. While the program is executing, the "Execute" menu in the Image window will be coloured green. When execution ends (as indicated by the "Execute" menu reverting to its usual black colour), the image window should be left holding the required image as specified by the supplied parameter values. You can then modify the parameter values, and re-execute the network (using the "Execution" menu again) to create a modified display.

12.1.2 Entering parameter values

Parameter values are displayed and entered using several "control panels", each panel containing controls related to a particular aspect of the demonstration. To access these control panels, open the "Windows" menu on the menu-bar of the Image window, and then select the "Open control panel by name" item. This will display a list of the control panels, and you can then open the panel by clicking on its name. The panel will remain open until you close it using the <**Close**> button at the bottom of the panel. Each panel has on-line help information describing its use which can be accessed by pressing the <**Help**> button at the bottom of the panel. Parameters can be set by selecting items from menus, pressing buttons, typing values into a box, etc, in a similar way to other Motif-based graphical user interfaces.

Always press return after typing a value into a data entry box.

12.1.3 Specifying the input data file

When run in this manner, the first thing which happens once the network has been loaded is that a pop-up window appears explaining how to get help on using the network. Click on the <**OK**> button within the pop-up window to get rid of this message. The next thing to do is to open the "Select input file..." control panel as described above, and supply an input file name. This can be done by typing the file name into the data entry box (remembering to press return). Alternatively you can browse through your file space by pressing the <**...**> button at the right hand end of the data entry box.

You must also specify the data format of the specified file using the <**Data format**> menu. The following formats are available:

DX native: These are files created by DX or by the "`$SX_DIR/ndf2dx`" application. They usually have a file extension of ".dx".

[5]This execution model (which is common to all DX networks) is somewhat different from many astronomical packages which allow parameter values to be specified *after* execution has commenced. For instance, a DX network will abort if the user has supplied a bad parameter value, where as other packages may re-prompt the use for a new value in such circumstances. Another more limiting feature is that DX networks cannot suggest run-time dynamic defaults for parameter values which are based on the results of some earlier processing of the data. It is sometimes necessary to hold this execution model in mind in order to understand the behaviour of the demonstration networks.

DX general: These are text files describing data typically produced by a user's own programs. The data itself can be either ASCII or binary, and may reside in another file referenced from within the supplied description file. See the DX *User's Guide* for details of the description file. Binary data written by a Fortran program cannot be read using this format.

DX general (Fortran binary): These are identical to "DX general" files, except that they refer to binary data created by a Fortran program, which consequently contains record control bytes in addition to the data bytes. The description file is identical to that for non-Fortran data.

NDF: These are standard Starlink NDF data structures.

Once you have specified the input file name and data format, you must execute the network as described above. This produces an image using default parameter settings. This is necessary because the network does not know the permitted range of the parameter values until it has examined the data.

12.1.4 Producing sequences of images

The demonstration networks can be re-executed automatically to produce a sequence of images. This is controlled using the sequencer control panel which can be opened by selecting "Sequencer" from the "Execute" menu in the Image window. This panel contains controls similar to a standard video recorder. To start the sequence press the play button (an arrow pointing to the right). To interupt it, press the stop button (a recessed square). The networks are written so that one or more parameters are incremented automatically each time it is executed by the sequencer. See the descriptions of the individual networks below for details of which parameters are controlled by the sequencer.

12.1.5 Changing your view of the displayed object and adding axes

In addition to the demonstration control panels, the display can also be modified using the "Options" menu situated on the menu-bar of the Image window. This menu allows control of (amongst other things) the size and position of the image, and the position in space from which the object is viewed. It also provides facilities for adding enumerated and labelled 3-d axes to the image.

View control: Select the "View control..." item from the "Options" menu. This produces a new window. You can press the "Set view" button to get a list of preset "view points" from which you can select, or you can press the "Mode" button, and then select "Rotate" to get a "virtual 3-d tracker-ball" which is controlled by pressing the left mouse button and moving the cursor over the image. You can also change from an *orthographic* view which is like viewing the object from a great distance with a powerful telescope, to a perspective view which is like being close up to the object. Note, the image is not re-drawn until you release the mouse button.

Image size and position: Select the "View control..." item from the "Options" menu, and then press the "Mode" button. Then select the "Pan/Zoom" item from the displayed list. Now position the cursor at the place where you want the new image centre to be. To zoom in, press the left mouse button and drag it until the displayed box encloses the area you are interested in, and then release the button. The image is re-drawn with the selected area filling the screen. To zoom out, press the right-hand mouse button instead. When the button is released, the image is re-drawn with the whole screen compressed into the selected area.

Axes control: Select the "Autoaxes..." item from the "Options" menu, and press the "Enabled" button at the top-left of the window which is then popped up. You can also enter labels for the axes using the three data entry boxes just below the "Enable" button. Other aspects of the axes can be controlled by pressing the "Expand" button at the bottom of the window. This displays more options which can then be set appropriately. Note, the axes do not appear until the program is re-executed, or the image window is reset by pressing <`control-F`> (which also resets the view to a default off-diagonal view).

Resetting the Image window: The Image window retains its setting even if you select a new input file. Of course, the old settings may not be appropriate for the new data (e.g. the new data may cover a much larger volume and so may not all fit in the Image window). Pressing `<control-F>` while in the Image window causes the view to reset to an off-diagonal display of size appropriate for the data.

12.1.6 Creating MPEG movies

MPEG movies may be created by saving a series of displayed images and then encoding them into an MPEG file. To do this, open the "MPEG control..." control panel as described above, and press the <**Save frames**> button. Each subsequently displayed image will be saved to disk as the next frame for the final movie. Remember to release this button if you do not want to save the next image in this way.

Once all frames have been saved, release the <**Save frames**> button, enter a file name in the <**MPEG file name**> data entry box (remembering to press return), press the <**Create MPEG**> button, and re-execute the network. The saved frames will be encoded into an MPEG file. Note, you must have the Berkeley `mpeg_encode` program installed for this to work.

12.2 "What do I do if..."

12.2.1 "...no image is displayed?"

This could be due to the "camera" being too far away or pointing in the wrong direction. Try resetting the camera by clicking on the title bar of the Image window, and then pressing `<control-F>`.

Sometimes there may be no data to display (for instance if you attempt to start a stream line at a place where there is no valid data). In this case, a window may pop up saying "No data to display". Try changing the parameter values and re-executing the network.

Another possibility is that the data is too faint to see. Try adjusting the colour controls for the network.

12.2.2 "...a big window containing lots of little boxes appears?"

If an error occurs while the network is executing (for instance if you specify an input file which does not exist or an illegal `Compute` expression) then an error message is written to the DX "Message window", and a window is opened showing the network with the offending module highlighted in orange. This is probably more than you really want! To get rid of the network editor window, select the "Close" item from the "File" menu at the left hand end of it's menu bar. You can also close the message window in a similar way. You can then change the parameter values and re-execute the network.

12.2.3 "...the whole thing seems to have locked solid?"

You can use the "Disconnect from server" and "Start server" items in the "Connection" menu of the Image window, to re-initialise the server process which does the numerical data processing for DX.

12.2.4 "...my sequence doesn't run?"

It is sometimes necessary to press the stop button (a recessed square icon) on the sequencer control panel, and then press the play button (an arrow pointing to the right) several times to get the sequence to play correctly.

12.3 Further details on using the iso network

This demonstration allows the user to import a data file, potentially containing several different scalar or vector quantities defined on a regular grid, and create a 3-d representation of a surface of constant value (an "iso-surface") in any one of the quantities in the file. Optionally, the colour at each point on the surface can be determined by the value of another quantity in the same input file. Alternatively, the surface can be given a uniform blue-grey colour. The opacity of the surface can be varied from completely transparent, to completely opaque. A sequence of different iso-surfaces (each corresponding to a different constant value) can be created and displayed automatically, and optionally saved in an MPEG animation file.

The following control panels are available:

Select input file...

This has been described in section 12.1.

MPEG control...

This has been described in section 12.1.

Select field to iso-surface...

Press the <**Field to define iso-surface**> button to see a menu of all the fields (i.e. defined quantities) in the input file. Select the field which you want to use to define the shape

of the iso-surface. If the input file only contains a single un-named field, then it will be indicated in the menu by the base file name.

You can display the common logarithm of the data instead of the actual data by pressing the <**Take common log of data**> button. The shape of the iso-surfaces will not change, but linearly spaced sequences of iso-surfaces will become logarithmically spaced. Negative or zero data values are excluded from the display.

Set iso-surface values...

The default data value for the iso-surface is the median value of the selected field. This can be over-ridden by entering a value in the <**Iso value**> box, and then pressing the <**Value selection method**> button and selecting "User-supplied value" from the menu.

Alternatively, an equally spaced sequence of values can be used in succession. The maximum and minimum values are entered in <**Max. iso value**> and <**Min. iso value**> and the number of steps is entered in <**No. of values**> . Executing the network will cause the Sequencer control panel to appear.

Surface colouring...

If you want the colour of the iso-surface to represent the data value in a second field (i.e. another quantity in the data file), then press the <**Use another field to colour the surface?**> button, and select the field by pressing the <**Field to use to colour surface**> button and selecting from the menu of available fields.

If the <**Use another field to colour the surface?**> button is *not* pressed, then the surface will have a uniform blue-grey colour.

The opacity value entered in the <**Surface opacity**> box applies to both sorts of colouring. A value of zero produces a completely clear (i.e. invisible) surface, and a value of one produces a completely opaque surface.

Press the <**Take common log of data**> button to use the logarithm of the data values to determine the colour. Zero or negative data values produce holes in the surface.

12.4 Further details on using the slice network

This demonstration allows the user to import a data file, potentially containing several different scalar or vector quantities defined on a regular grid, and create a representation of a 2-d slice through the grid perpendicular to one of the axes. The colour at each point on the slice is determined by the magnitude of the data value, or optionally of the common logarithm of the data value. The orientation and position of the slice can be set by the user, and a bounding box can be displayed encompassing the whole volume. A sequence of equally spaced slices can be created and displayed automatically. Images can be saved and encoded into an MPEG animation.

The following control panels are available:

Select input file...

This has been described in section 12.1.

MPEG control...

This has been described in section 12.1.

Select field to display...

Press the <**Field to display**> button to see a menu of all the fields (i.e. defined quantities) in the input file. Select the field which you want to display. If the input file only contains a single un-named field, then it will be indicated in the menu by the base file name.

You can display the common logarithm of the data instead of the actual data by pressing the <**Take common log of data**> button. Negative or zero data values are excluded from the display.

Set slice position...

The <**Show bounding box**> button determines if a bounding box should be displayed with the slice. The bounding box encloses the entire data set.

The <**Axis name**> button allows you to select the orientation of the slice. The displayed slice will be perpendicular to the selected axis.

The position at which the slice cuts the axis can be determined in four ways, selected using the <**Position selection method**> menu:

Default position: A default position is used. This is the mid point of the selected axis.

User-supplied position: The user-supplied position is used. This should be entered in the <**Axis value**> box.

Sequence of positions: An equally spaced sequence of positions is used in succession. The maximum and minimum values are entered in <**Max. axis value**> and <**Min. axis value**> , and the number of steps is entered in <**No. of values**> . Executing the network will cause the Sequencer control panel to appear.

Image window probe: A cursor is used to set the position. Open the view control dialogue box by clicking on "View Control" in the "Options" menu of the Image window. Then click on "Mode" and select "Cursors" from the displayed list of options. Position the arrow cursor somewhere over the displayed image and double click. This will produce a small square cursor (or *probe*) which can be dragged around by pointing at it and holding the left mouse button down.

The probe is constrained to move parallel to an axis, and the projection of the probe position on to each face of the enclosing box is displayed in order to ease the problem of positioning the probe in 3 dimensions. Once the probe is positioned, release the left mouse button and execute the network.

Slice colouring...

The slice will be displayed with a colour table going from blue at low data values to red at high data values. The data values corresponding to pure red and blue can either be entered by the user, or default values can be used. The method is determined by the setting of the <**Method**> button. Any data values which fall outside the given limits will produce holes in the slice (i.e. the background colour, black by default, will show through). The default values are the maximum and minimum data values on the slice. Thus the default scaling will change as the slice is moved through the data volume.

This panel also allows you to change the size of the labels on the colour bar displayed at the right hand edge of the window. The default size is multiplied by the supplied factor. The default size is dependent on the distance between tick marks and so will in general alter if the data range covered by the colour bar is changed.

12.5 Further details on using the stream network

This demonstration allows the user to import a data file, potentially containing several different quantities defined on a regular grid, and create a representation of a vector field using stream lines.

The starting positions of the stream lines can be given explicitly by the user, using cursor or keyboard. Alternatively, stream lines can be started at each point on a regular grid covering a specified volume. These positions may then be modified using an arbitrary vector algebra expression, involving the original positions and the frame number in a sequence of frames. Thus sequences of images can be made in which the stream line starting positions move through the data.

The stream lines may be coloured to represent the magnitude of the field, or of another field in the same data file. An image may also contain an iso-surface defined by a separate field in the input data file, and a bounding box encompassing the whole volume. The point from which the objects are viewed can be automatically rotated in 3-D between successive frames to give a continuous rotation. Any set of displayed images can be saved and encoded into an MPEG animation.

The following control panels are available:

Select input file...

This has been described in section 12.1.

MPEG control...

This has been described in section 12.1.

Streamline positions...

This panel controls the number of stream lines displayed, their starting positions and maximum lengths. The stream lines are defined by the field selected with the <**Field to define stream lines**> button, and this field must contain a vector quantity.

The stream line starting positions can be specified in three ways, as selected by the <**Method**> button:

Keyboard: The positions are entered using the controls at the lower left of the panel, under the heading "Keyboard positions". To add a new position, enter the x, y and z co-ordinates in the three slider boxes at the bottom, and then press <**Add**> . To delete a position, click on the line holding the position in the panel just above the <**Add**> and <**Delete**> buttons, and then press <**Delete**> .

Grid: An evenly spaced grid of positions is used as defined by the controls at the lower right of the panel. The volume enclosing the grid is defined with <**Grid lower bounds**> and <**Grid upper bounds**> , and the number of positions along each axis is defined with <**No. of grid points**> .

Probes: Positions are given using the cursor to control probes in the image window. Press "View control" in the "Options" menu of the Image window. Then select "Cursors" using the "Mode" button in the view control dialogue box. Position the arrow cursor somewhere over the image and double click on the left button. This should produce a small square probe which can be dragged around the 3-d volume represented by

the displayed image. To do this point at the probe with the cursor, press the left mouse button and at the same time move the mouse. Note, you can only move the probe parallel to one of the axes at any one time. Spots are shown on a bounding box indicating the projection of the probe position onto the faces of the bounding box. Many probes may be introduced into the volume in this way. To delete a probe, point at it with the cursor and double click on the left mouse button again. When all the required probes have been positioned, re-execute the network to create the corresponding stream lines.

Once the positions have been defined by one of the above methods, they can be modified by entering an algebraic expression into the <**Expression**> box. This is an expression suitable for use by the `Compute` module (see the DX *User's Reference Guide* for details), in which the symbol p represents the original positions. For instance the expression "$p + [0, 1, 0]$" increments all the positions by 1 in the y direction. The default expression "p" leaves the positions unchanged.

The symbol i can be used to represent the frame number within a sequence, starting at 1. For instance, the expression "$p + [i - 1, 0, 0]$" increments the stream line starting positions by 1 along the x axis for each frame in the sequence. The number of frames in the sequence is set in the <**No. of frames**> box. A sequence of more than one frame can be controlled using the Sequencer control panel, which should automatically appear if <**No. of frames**> is larger than 1. A sequence of 1 frame isn't really a sequence at all and is created as normal using the "Execute" menu.

Stream lines end when they pass outside the bounding box, reach a point where the velocity has zero magnitude, or exceed the "time" given in the <**Max. time**> box. This presumes that the vector represents a velocity, and that a point moving with unit speed (i.e. of unit vector magnitude) moves a unit distance (as defined by the spatial co-ordinate system of the data) in unit time.

The units of time are defined by the data (i.e. a point with unit vector magnitude moves a unit distance in unit time).

Display control...

A yellow box is displayed which encloses the entire data volume if the <**Display bounding box**> button is pressed.

The size of the colour bar annotation can be changed by entering a value in the <**Scale for colour bar labels**> box. The default size is multiplied by this factor. The default size depends on the distance between tick marks and so varies with the data range. Note, a colour bar is only displayed if the stream lines are coloured to show some data value (see the "Streamline colours..." control panel).

The stream lines can be shown as thin lines, tubes or ribbons, and the <**Rendering style**> button is used to select the method to use. Ribbons can show twist in the vector field.

Streamline colours...

If the <**Ignore colour data**> button is pressed, the stream lines are coloured a uniform yellow colour irrespective of the settings of the other controls on this panel. Otherwise the quantity selected using the <**Field holding colour data**> button is used to determine the colour at each point of the streamline.

If the <**Use default scaling for colour data**> button is pressed, the colour scaling is set so that the minimum data value along any of the displayed stream lines is shown as blue, and the maximum value is shown as red.

If the <**Use default scaling for colour data**> button is *not* lit, the data values entered by the user in the two boxes at the bottom of the panel are used to define blue and red. Any data values outside this range will be shown as gaps in the stream lines.

If the specified field is a vector quantity then the magnitude of the vector is used to determine the colour. The common logarithm of the data is used to define the colour instead of the data itself if the <**Use common log of colour data**> button is pressed.

Iso-surface control...

If the <**Display iso-surface**> button is pressed, an iso-surface is displayed together with the stream lines. The iso-surface shows a surface of constant value in the quantity selected using the <**Field to define iso-surface**> button, which may be a scalar or vector quantity.

If the <**Use default iso-surface value**> button is pressed, the median data value is used to define the iso-surface. Otherwise, the data value entered in the <**Iso-surface data value**> box is used. For vector fields this should be the vector magnitude on the surface.

The surface has a uniform blue-grey colour, and it's opacity can be set using the <**Iso-surface opacity**> data entry box. A value of 0.0 produces a completely clear (i.e. invisible) surface, and a value 1.0 produces a completely opaque surface.

Auto rotation...

If the <**Enable auto-rotation**> button is pressed, each displayed image is rotated in 3-D by the number of degrees given in the <**Rotation speed**> box relative to the previous image. If the <**Reset**> button is pressed the view point resets to the original view point.

The rotation is around a great circle centred on the to-point of the current camera. This is usually the centre of the object, but this can be changed using the view control facilities of the Image window.

12.6 Further details on using the scatter network

This demonstration allows the user to import and examine scattered particle data. Each particle is defined by a position in 3-d space and any number of additional data quantities, which may be vectors or scalars.

Images are created which allow the user to view an arbitrary subset of the particles from any point in space. Each particle may be presented by a *glyph* consisting of a simple point, or a sphere (rendered in 3-D), or a 3-d rocket-like object (useful for vector quantities). The size and shape of each sphere or rocket can be determined by any arbitrary combination of the particle's data values, its position, its distance from the camera, etc. The colour used to represent each particle can also be determined independently in a similar arbitrary manner. Objects used to represent particles may be made partially transparent so that crowded areas appear brighter than sparsely populated areas.

A series of data sets can be imported and displayed automatically, and selected particles can be followed from one frame to the next. The scattered data can be binned or interpolated onto a regular grid which can then be saved on disk for visualisation using other demonstration networks. The point from which the objects are viewed can be automatically rotated in 3-D

between successive frames to give a continuous rotation. Any set of displayed images can be saved and encoded into an MPEG animation. A bounding box can optionally be displayed.

*Note, the view control facilities of the Image window are not available with this network unless the camera update mode is set to manual on the "*Camera control...*" panel.*

The following control panels are available:

Select input file...

This has been described in section 12.1. In addition, if you want to look at a sequence of files, you can include the string "%d" in the file name. This string will be replaced by the current frame number returned by the sequencer. This number starts at 1 and increases up to the value given in the <**Max. frame number**> data entry box. To start the sequence playing, select "Sequencer" from the "Execute" menu on the Image window. This will pop up a new window containing controls for the sequencer. These controls are similar to those for a standard video recorder. Pressing the play button (an arrow pointing to the right) will start the sequence.

MPEG control...

This has been described in section 12.1.

Camera control...

This panel controls the projection between 3-d space and the 2-d screen. This projection is determined by the current *camera*. The camera can be controlled in several ways, determined by the setting of the <**Camera update mode**> button:

Auto: This is the default. The camera is chosen automatically to suit the displayed object. This will be an off-diagonal (if the data is 3-d) orthographic view from a distance such that the object fills most of the screen. Each time the object changes, the camera is updated automatically. Note, in this mode the view control facilities provided by the Image window are inoperative.

Manual: The camera is not updated automatically, but retains its parameters even if the object is changed. This mode allows the user to change the camera manually using the view control facilities of the Image window.

Freeze: The camera is frozen in its current state. The view remains the same even if the viewed object changes, and the view control facilities of the image window are inoperative.

Auto-rotate: The view point is rotated in 3-D by the number of degrees given in the <**Rotation speed**> data entry box each time the network is executed, but the camera is otherwise frozen. The view control facilities of the Image window are inoperative.

Define data quantities...

Several aspects of the displayed image (such as the colour and size of the glyphs used to represent each particle) can be controlled by the values stored in the imported data file. Up to three different quantities can be used in this way, and these quantities are referred to as a, b and c within the demonstration. The correspondence between these names and the quantities in the data file can be specified using the buttons on this control panel.

Select positions to display...

Particles may be excluded from the displayed image by entering a string in the <**Selection expression**> data entry box (remember to press return after you have typed in the string). This string is an algebraic expression which is passed to the `Compute` module for evaluation (see the DX *User's Reference Guide* for a description of the syntax). It is evaluated for each particle, and only those particle for which the resulting value is greater than zero are displayed.

The expression may refer to any of the following variables:

a, b, c – These are the imported data values specified on the "Define data quantities..." control panel. They may be vector or scalar.

p – This is the position of the current particle, padded with trailing zeros if necessary to make it 3-d. It is a 3-d vector.

v – This is the position in 3-d space from which the object is being viewed. It is a 3-d vector.

f – This is the current frame number determined by the sequencer. It is a scalar.

e – This is the offset of the current particle from the start of the list of particles. The first particle in the data file has $e = 0$, the second has $e = 1$, etc. It is a scalar.

Here are some example expressions:

"1.0" – gives every particle the value of 1.0. Since this is greater than zero, every particle is displayed. This is the default.

"0.0" – causes no particles to be displayed.

"$a - b$" – causes particles to be displayed only if the value of the a quantity is greater than the b quantity. These quantities are defined on the "Define data quantities..." control panel.

"$e < 3$" – causes only the first 3 particles to be displayed.

"$p.x > 10$" – causes only particles with x co-ordinates larger than 10 to be displayed.

"$mag(p - v) > 10$ && $a < 0$" – displays particles which are further than 10 units away from the observers view point (in the spatial co-ordinate system of the data), and which also have an a value less than zero.

Particles may also be selected for display using the view control facilities of the Image window. To do this you must first select manual camera update mode on the "Camera control..." control panel. The view point may then be adjusted manually by panning, zooming, navigating through the data space, etc, until the view contains only the positions in which you are interested. Then press <**Exclude non-visible particles**> and re-execute the network. This will exclude all particles which do not currently fall within the bounds of the image window. You will normally then press the <**Freeze particle selection**> in order to stop particles being re-selected as the view changes. You can then change the view-point, and only the selected particles will remain in the image.

Pressing the <**Freeze particle selection**> button causes the current selection of particles to be frozen. This is useful, for instance, if you want to follow selected particle through a sequence of input files. If you don't freeze the selection, different particles may be displayed

from each input file (for instance if the selection expression depends on the data quantities in the file).

Pressing the <**Show no. of selected particles**> button causes a window to pop up telling you the number of selected particles.

Display control...

This panel controls several extra objects which can be included in the display along with the particle data.

The <**Bounding box**> button determines if a bounding box is to be displayed. If it is set to "Current" then a box which just encompasses the currently selected positions is displayed. This box is updated as the positions change. If it is set to "Freeze" then the box is not updated as the positions change. If it is set to "None" then no bounding box is displayed.

Selecting "Yes" from the <**Show colour bar**> menu causes a colour bar to be included showing the colour value associated with each colour (as set by the "Colour control..." panel). The scale of the labels on the colour bar may be changed by entering a value in the <**Colour bar label scale**> box.

Pressing the <**Show file name caption**> button causes a caption showing the current input file can be added to the display.

Colour control...

The colour used to represent each selected particle is determined by a two-stage process:

1. a scalar value known as the *colour value* is associated with each particle.
2. these colour values are mapped onto actual colours using one of two standard colour look-up tables (one gives a grey-scale image and the other gives a colour image).

In addition, the objects used to represent each particle may be made partially transparent by setting an opacity value of less than 1.0. This gives some idea of the density of particles (crowded areas with many overlapping particles will look more opaque than sparsely populated areas).

The colour value for each particle is determined by a text string entered in the <**Colour value expression**> data entry box which is an algebraic expression passed to the `Compute` module for evaluation. It is evaluated for each particle, and gives the colour value. If the supplied expression produces vector values, then the modulus of the vector is used as the colour value. See the description of the "Select positions to display..." control panel for the variables which may be included in this expression.

Here are some example expressions:

"1.0" – causes all particles to have a constant colour value of 1.0. This will result in all particles having the same colour (as determined by the colour look-up table). This is the default.

"a" – causes each particle's colour value to be equal to the a quantity defined on the "Define data quantities..." control panel.

"$sin(a * b)$" – causes each particle's colour value to be the sine of the product of the a and b quantities as defined in the "Define data quantities..." control panel.

"$1.0/mag(p-v)**$**2**"** – causes each particle's colour value to be inversely proportional to the square of the distance to the observer. This is useful for getting an idea of the global structure of the data set, especially when combined with the rotation facilities of the "Camera control..." control panel and the "View control" options of the Image window.

In addition, a constant colour value may be assigned to all currently visible particles (i.e. particles which are within the bounds of the Image window) by pressing the <**Use specified constant colour value for visible particles**> button. The value to use is entered in the <**Colour value for visible particles**> data entry box. All particles which are outside the bounds of the Image window retain the colour values established by the <**Colour value expression**> box. This facility can be useful for following selected particles between successive frames in a sequence. You would typically display the first frame, and then use the "View control" options of the Image window to adjust the view so that only the particles of interest are visible. You then press the <**Use specified constant colour value for visible particles**> button, and re-execute the network. This will assign the given colour value to the visible particles. You would normally then press the <**Freeze colour values**> button (so that these colour values are not updated each time the view changes) and then change the view to show a larger volume.

If the <**Show colour value statistics**> button is pressed, a pop-up window is created showing the mean, standard deviation, maximum, minimum and median colour values for the current frame.

Having assigned a colour value to each particle, these are mapped into actual colours using either a grey-scale or colour look-up table, as selected by the <**Colour table**> button. The grey-scale table uses black to represent the minimum colour value, and white for the maximum colour value, with shades of grey in-between. The colour table uses blue and red instead of black and white, with smoothly varying colours in-between. A bar giving a key to the colour used for each colour value can be displayed using the "Display control..." control panel. The maximum and minimum colour values used by the table can either be specified by the user (in which case the colour table remains fixed for all frames), or default values can be used which covers the entire range of the colour values of the current frame (in this case the colour table may change between frames). This choice is made using the <**Colour table range**> button. In "User-supplied" mode, the user enters values into the <**Colour table minimum**> and <**Colour table maximum**> data entry boxes.

Glyph control...

Each particle may be represented by either a point, or a *glyph*, as selected by the <**Marker type**> button. Glyphs are 3-d objects which are rendered to show light and shade, and which get smaller as they recede from the observer as a normal 3-d object would do (if a perspective camera is being used). Spheres are used to represent a scalar value at a given position, and rocket-like structures are used to represent a vector value at a given position. Points are single screen pixels which do not have a 3-d appearance.

The size of the glyph for each particle is defined using the string entered in the <**Glyph size expression**> data entry box. This string is an algebraic expression which is passed to the `Compute` module for evaluation. It is evaluated for each particle and gives the size of the associated glyph. If the expression evaluates to a scalar quantity then spherical glyphs are used with radius given by the expression. If the expression evaluates to a vector quantity then rocket glyphs are used with length given by the expression. See the description of

the "Select positions to display..." control panel for the variables which may be included in this expression.

Here are some example expressions:

"1.0" – causes all particles to be displayed with the same sized spherical glyph. This is the default.

"a" – causes all particles to be displayed as a glyph with size (and direction if a is a vector) given by the a quantity.

"$[a, b, c]$" – causes particles to be represented as rocket glyphs with direction and length determined by the three quantities a, b and c which should be scalar values.

The resulting glyph sizes are measured in the data's spatial co-ordinate system, and may be modified by a constant scale factor by entering the factor in the <**Glyph scale factor**> data entry box, and ensuring that the <**Use default glyph scale**> button is not pressed. If this button *is* pressed, the value in the <**Glyph scale factor**> box is ignored and a default value is used which attempts to avoid either very large or very small glyphs. The thickness of rocket glyphs can be modified by entering a factor in the <**Glyph thickness**> box.

A constant glyph size can be assigned to all particles which are currently within the bounds of the Image window by entering the value in the <**Constant size for visible glyphs**> box, pressing the <**Use constant size for visible glyphs**> button and re-executing the network. You can press the <**Freeze glyph sizes**> button if you want to retain the current glyph sizes between successive frames.

Export gridded data...

If the <**Save gridded data**> button is pressed, then the data specified by the <**Data to grid**> button is converted to a regular grid, and exported to a disk file in native DX format. The button is automatically reset.

The name of the disk file is given in the <**Output filename**> box. The supplied string may contain a single occurrence of the sub-string "%d" which will be replaced with the current frame number (see the "Select input file..." control panel).

The <**Data to grid**> button determines which data is exported. Selecting "Selected data" causes only the positions selected using the "Select positions to display..." control panel to be exported. Selecting "Input data" causes the entire input data file to be exported.

The grid onto which the scattered data is mapped encompasses the exported data, and each cell is cubic with dimension given by <**Grid cell dimension**> . If zero is given for the grid cell dimension, then a replacement value is chosen which gives the required number of cells along the x axis, as entered in the <**X axis cell count**> box.

The data values stored at each output grid point are determined by the setting of the <**Data to grid**> button and the <**Output data**> button:

- If <**Output data**> is set to "Density values", then each output grid point holds a single scalar value which is an estimate of the density of particles in the neighbourhood of the grid point.

- If <**Output data**> is set to "Mean data values" and <**Data to grid**> is set to "Selected data", then each grid point contains a single data value which is an estimate of the mean glyph size value of the particles in the neighbourhood of the grid point (see control panel "Glyph control...").
- If <**Output data**> is set to "Mean data values" and <**Data to grid**> is set to "Input data", then each grid point has the same number of data values as each input particle, and each output data value is an estimate of the mean of the corresponding input data value in the neighbourhood of the output grid point.

There are two ways of obtaining these estimates, selected by the <**Gridding method**> button:

Sampling: The input data is interpolated at the positions of the output grid points. For each output grid point, the nearest N input particles are identified (where N is given by the value in the <**No. of neighbours to use**> box). The output data values are either the mean of these input data values (if <**Output data**> is set to "Mean data values"), or an estimate of the density of particles in the neighbourhood (if <**Output data**> is set to "Density values"). The output data are *position-dependent*. Note, this method is very slow. Progress reports are given in the DX message window showing how far the re-gridding has to go.

Binning: The input particles are binned into the output grid. Each output grid cell contains either the mean of the particle data values in the cell, or the density of particles in the cell (dependent on the setting of <**Output data**>). The output data are *connection-dependent*.

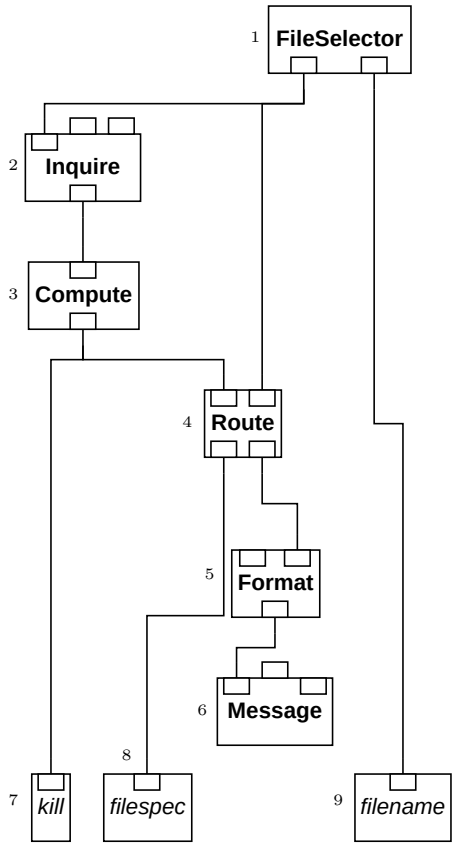
12.7 A detailed look inside the iso demonstration network

The SX demonstration networks attempt to be reasonably general in terms of the data they can be used with and the options they provide. This necessarily causes them to be more complex than networks written to perform specific visualisations of particular data files. For instance, displaying a simple iso-surface can be done using just three modules (`Import`, `Isosurface`, and `Image`), whereas the "iso" demonstration contains 108 modules! Generality is bought at the expense of much greater complexity, and this complexity results in there being many more things to go wrong! It is usually a good idea to make your networks as specific as possible.

By their nature, networks are difficult to structure and document. In an attempt to ease this problem, each network has been divided into several sections, which (to a greater or lesser degree) perform separate, well-defined tasks. If you examine the networks using DX, you will find these sections placed side by side, working from left to right in the network editor window.

To illustrate the way that these networks are written, the "iso" network is described in detail in the following pages. Each section is presented graphically, with a description of what the section does on the following page.

Demonstration “iso” - section 1



Demonstration "iso" - section 1

Purpose:

Issue initial help and obtain the input file.

Description:

If the network is executed before an input file name has been given, a pop-up window containing help information is displayed, and the rest of the network is prevented from executing. If an input file name *has* been given, the full file specification and the file base name are passed on to the rest of the network, which then executes normally.

Modules:

1 – `FileSelector`: This module returns the file name entered by the user in the "Input file name:" widget in the "Select input file..." control panel. The left output is the full file specification and the right output is the file base name. If no file has been given, then a `NULL` value is returned for both outputs.

2 – `Inquire`: Uses the "is null" enquiry to see if a `NULL` file specification was returned by module 1 (`FileSelector`). The output is set to 1 if this is the case (i.e. no input file was specified), and zero otherwise.

3 – `Compute`: Adds 1 onto the output from module 2 (`Inquire`) putting it in the range [1,2], suitable for use by module 4 (`Route`).

4 – `Route`: Routes execution to the left output if module 3 (`Compute`) outputs 1, or to the right output if it outputs 2. The file specification obtained by module 1 (`FileSelector`) is passed on to the selected output. The result of this is that if a file name has been given, then it is passed on to module 8 (*filespec*). If no file name has been given, then a `NULL` value is passed on to module 5 (`Format`). None of the modules which are dependent on the un-selected output are executed, with the exception of any `Collect` modules (which are always executed).

5 – `Format`: Creates a string holding the help message. This module is only executed if module 3 (`Compute`) outputs 2 (i.e. if no data file has been given). The `NULL` value routed to its right input is used only to trigger execution of the module. It does not form part of the string, which is permanently assigned to its left input (the **template** parameter) using the module's configuration dialogue box.

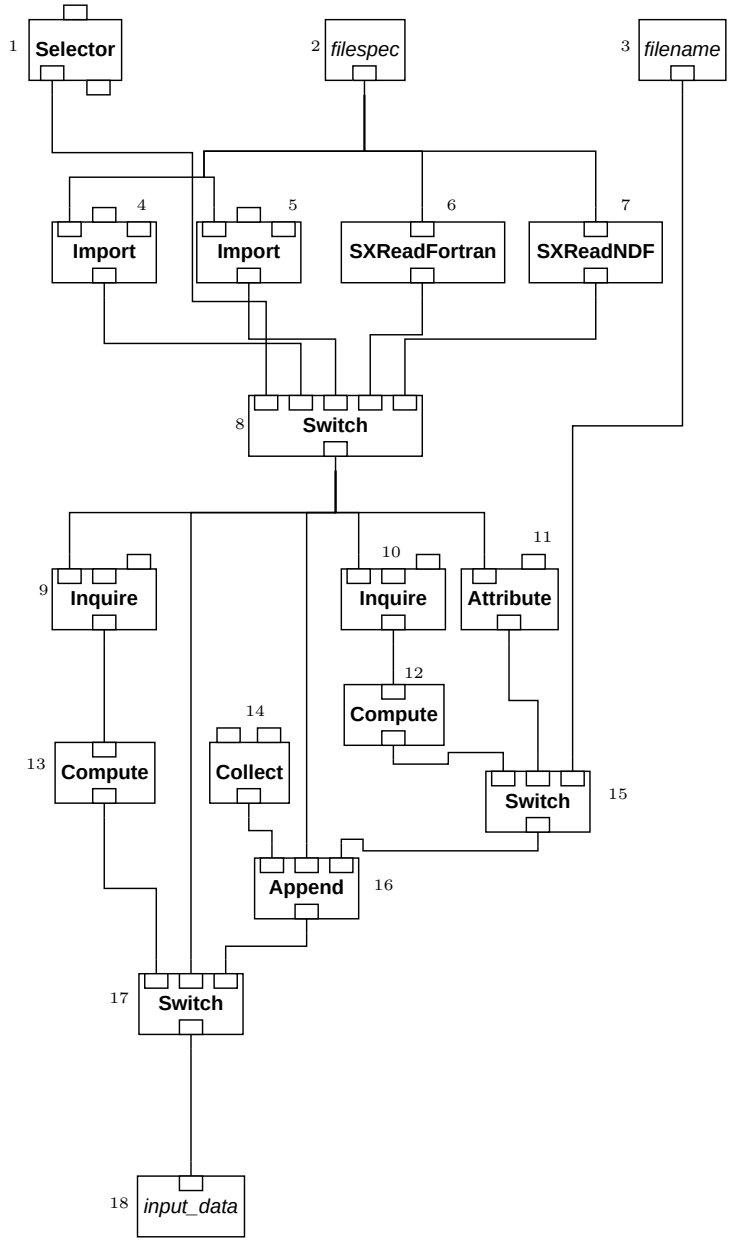
6 – `Message`: Creates a pop-up window containing the string output by module 5 (`Format`). Execution of the network stops when this has been done (except for any `Collect` modules in other sections).

7 – *kill*: This is a `Transmitter` module to which the name "*kill*" has been given using its configuration dialogue box. It transmits the value returned by module 3 (`Compute`) to corresponding `Receiver` modules in other sections of the network. This value is used to explicitly stop execution of (or "kill") modules down-stream of any `Collect` modules, if no input file has been given. This is necessary because the `Collect` module is always executed, even if none of its inputs have been executed.

8 – *filespec*: This is a `Transmitter` module to which the name "*filespec*" has been given. It transmits the full file specification supplied by the user to corresponding `Receiver` modules in other sections of the network.

9 – *filename*: This is a `Transmitter` module which transmits the base file name to corresponding `Receiver` modules in other sections of the network.

Demonstration "iso" - section 2



Demonstration "iso" - section 2

Purpose:

Import the data and ensure it is a group structure.

Description:

The data in the input file is imported, using a module appropriate for the file's format. The imported data may be structured in several ways. For instance, it may contain several quantities (or "fields" in DX parlance) in a "group" structure, or it may contain just a single quantity. If it contains only a single quantity, the field may or may not have an associated name. The modules down-stream of module 8 (`Switch`) ensure that the data is passed on to the rest of the network in a consistent way so that further checks on the data structure can be avoided. If the input file contains only a single field, a new empty group structure is created and the field is put in it. If the field has no name it is given a name (the base file name). The rest of the network can then assume that the data is a group of named fields.

Modules:

1 – `Selector`: This module returns an integer in the range [1,4] identifying the data format selected by the user using the "Data format" widget in the "Select input file..." control panel.

2 – *filespec*: A `Receiver` module which picks up and returns the "*filespec*" value transmitted by section 1 of the network. This value is the full specification of the input data file.

3 – *filename*: Returns the base file name transmitted by section 1 of the network.

4 – `Import`: Imports the data assuming it is in DX "native" format.

5 – `Import`: Imports the data assuming it is in DX "general" format.

6 – `SXReadFortran`: Imports the data assuming it is binary data created by a Fortran program, in DX "general" format.

7 – `SXReadNDF`: Imports the data assuming it is a Starlink NDF.

8 – `Switch`: Routes the output from the required data reader through to the rest of the network. Un-required inputs are not calculated.

9 – `Inquire`: Inquires if the input data is a group of fields.

10 – `Inquire`: Inquires if the input data (assuming it is a single field) has a name.

11 – `Attribute`: Obtains the name of the input field (assuming it has one).

12 – `Compute`: Adds 1 to the output from module 10 (`Inquire`) so that it is in the range [1,2] and can thus be used to select one of the two inputs to module 15 (`Switch`).

13 – `Compute`: Adds 1 to the output from module 9 (`Inquire`) so that it is in the range [1,2] and can thus be used to select one of the two inputs to module 17 (`Switch`).

14 – `Collect`: Creates an empty group to which the input field can be appended.

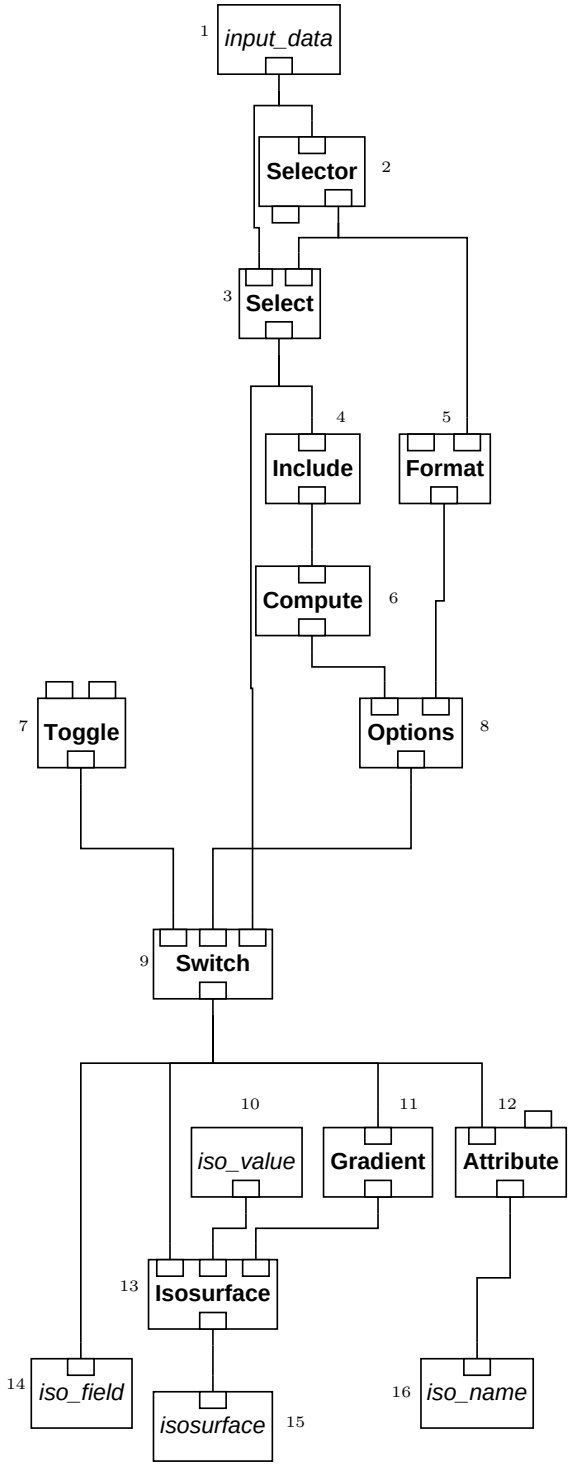
15 – `Switch`: Passes through the name of the input field (if it has one), or the base name of the input file otherwise.

16 – `Append`: Puts the input field into a group, giving it the name passed on by module 15 (`Switch`).

17 – `Switch`: If the input data is a group of fields, it is passed through to the output. If the input data is a single field, the newly created group containing the input data is passed through to the output.

18 – *input_data*: Transmits the input data in the format assumed by the rest of the network.

Demonstration “iso” - section 3



Demonstration "iso" - section 3

Purpose:

Create the iso-surface.

Description:

The user specifies which of the fields in the input file is to be used to define the iso-surface. This field is extracted, and optionally replaced by a field holding the common logarithm of the data values. The iso-surface is then created and passed on to later sections of the network.

Modules:

1 – *input_data*: Returns the input data transmitted by section 2 of the network. This will always be a group of named fields.

2 – `Selector`: Returns the name of the field to be used to define the iso-surface. This is specified by the user on the "Select field to iso-surface" control panel. The module automatically reads the names of all the fields in the input data group and supplies them as options on the control panel.

3 – `Select`: Extracts the selected field from the input data group.

4 – `Include`: Removes all zero or negative data values from the selected field, so that the logarithm of the data can safely be taken.

5 – `Format`: Creates a string of the form "`Log(<name>)`" where `<name>` is the name of the field.

6 – `Compute`: Takes the common logarithm of the data.

7 – `Toggle`: Returns 1 if the user has pressed the "Take common log of data" button on the "Select field to iso-surface" control panel, and 2 otherwise.

8 – `Options`: Assigns the string created by module 5 (`Format`) as the new name of the field.

9 – `Switch`: Passes on either the original data, or the logarithm of the data, as determined by module 7 (`Toggle`).

10 – *iso_value*: Receives the data value which is to be used to define the iso-surface. This is transmitted by section 6 of the network.

11 – `Gradient`: Calculates the gradient of the data. `IsoSurface` has an option to calculate the gradient itself. But it is more efficient to calculate it once using the `Gradient` module, than to allow `IsoSurface` to re-calculate it every time a new iso-surface is required.

12 – `Attribute`: Obtains the name of the data field. This may have been modified to indicate that the logarithm of the data has been taken.

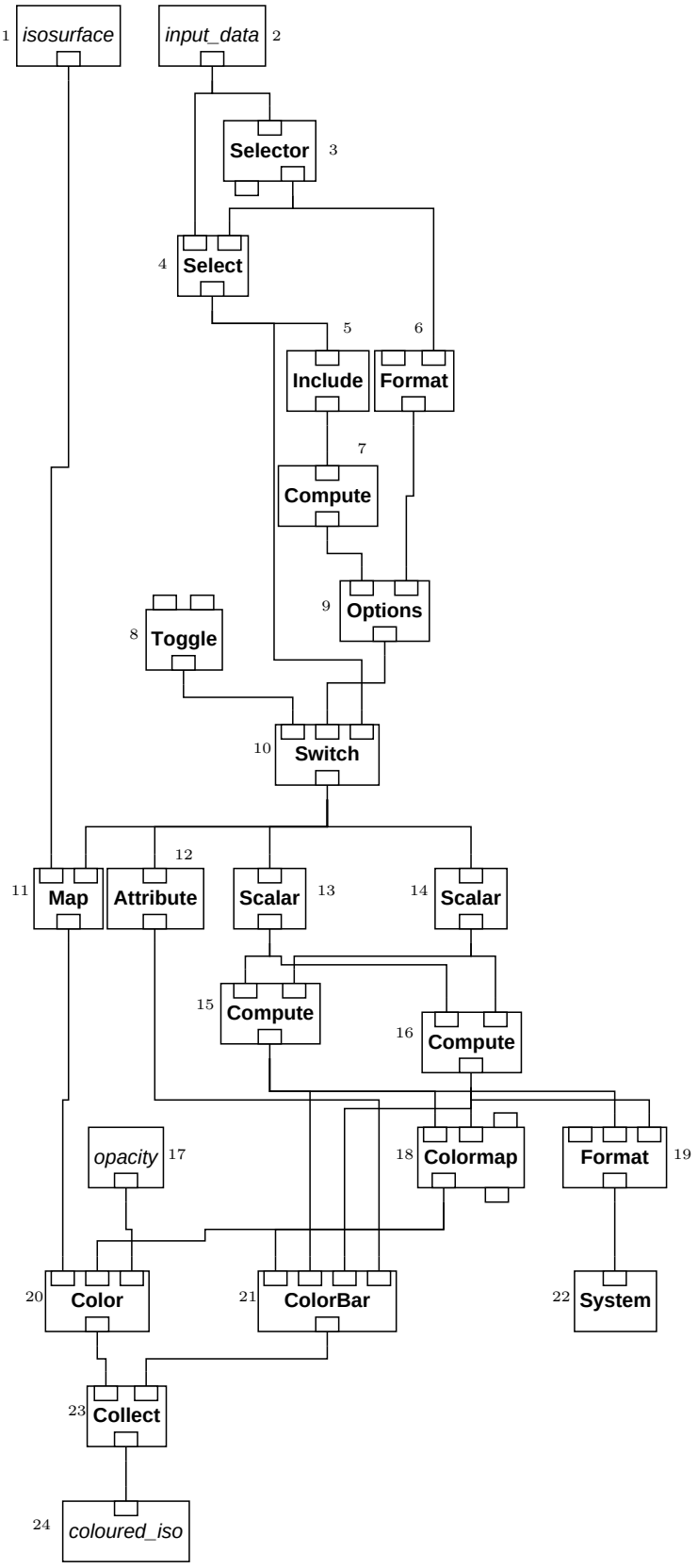
13 – `IsoSurface`: Creates the iso-surface structure. Its inputs are the data array, the data value to define the iso-surface, and the gradient of the data array.

14 – *iso_field*: Transmits the field which was used to define the iso-surface for later use.

15 – *isosurface*: Transmits the iso-surface to the rest of the network.

16 – *iso_name*: Transmits the name of the field used to define the iso-surface for later use.

Demonstration "iso" - section 4


1 isosurface
input_data 2
Selector 3
4 Select
5
6
Include
Format
7
Compute
9 Options
8 Toggle
10 Switch
12
11 Map
Attribute
Scalar 13
14 Scalar
15 Compute
16 Compute
opacity 17
18 Colormap
Format 19
20 Color
21 ColorBar
22 System
23 Collect
24 coloured_iso

Demonstration "iso" - section 4

Purpose:

Colour the iso-surface created by section 3.

Description:

This section colours the iso-surface using the data value (or optionally the logarithm of the data value) in another specified field to determine the colour at each point. The iso-surface is also given a constant opacity. If the user requests a constant colour for the iso-surface, then the output from this section (i.e. from module 24) is not needed, and so none of this section is executed (except for those parts which are needed to execute the `System` module 22).

Modules:

1 – *isosurface*: Receives the geometry field describing the iso-surface, transmitted by section 3.

2 – *input_data*: Receives the input data group, transmitted by section 2.

3 – `Selector`: Obtains the name of the field to be used to colour the iso-surface. This is specified by the user on the "Surface colouring..." control panel.

4 – `Select`: Extracts the selected field from the input data group.

5 – `Include`: Removes all zero or negative data values from the selected field.

6 – `Format`: Creates a string of the form "`Log(<field name>)`".

7 – `Compute`: Takes the common logarithm of the data.

8 – `Toggle`: Returns 1 if the user has pressed the "Take common log of data" button on the "Surface colouring..." control panel, and 2 otherwise.

9 – `Options`: Assigns the string created by module 6 (`Format`) as the new name of the field.

10 – `Switch`: Passes on either the original data, or the logarithm of the data.

11 – `Map`: For each position on the iso-surface, the data value at the corresponding position in the field passed on by module 10 (`Switch`) is found. These data values, together with the iso-surface positions, are passed on to module 20 (`Color`).

12 – `Attribute`: The name of the field defining the iso-surface colour is obtained.

13 – `Scalar`: Obtains the minimum data value to be displayed, as entered by the user in the "Surface colouring..." control panel. The value is constrained to be in the range of the field passed on by module 10 (`Switch`).

14 – `Scalar`: Obtains the maximum data value to be displayed, as entered by the user in the "Surface colouring..." control panel.

15 – `Compute`: Returns the minimum of the values passed on by modules 13 and 14 (`Scalar`). This is done in case the user gave the values the wrong way round. The returned value is displayed as blue.

16 – `Compute`: Returns the maximum of the values passed on by modules 13 and 14 (`Scalar`). The returned value is displayed as red.

17 – *opacity*: Receives the opacity to be given to the iso-surface, transmitted by section 5.

18 – `Colormap`: Creates a colour map which maps the supplied minimum and maximum data values on to blue and red.

19 – `Format`: Formats a blank string. See module 22 (`System`).

20 – `Color`: Associates a colour and opacity with each of the mapped data values on the iso-surface.

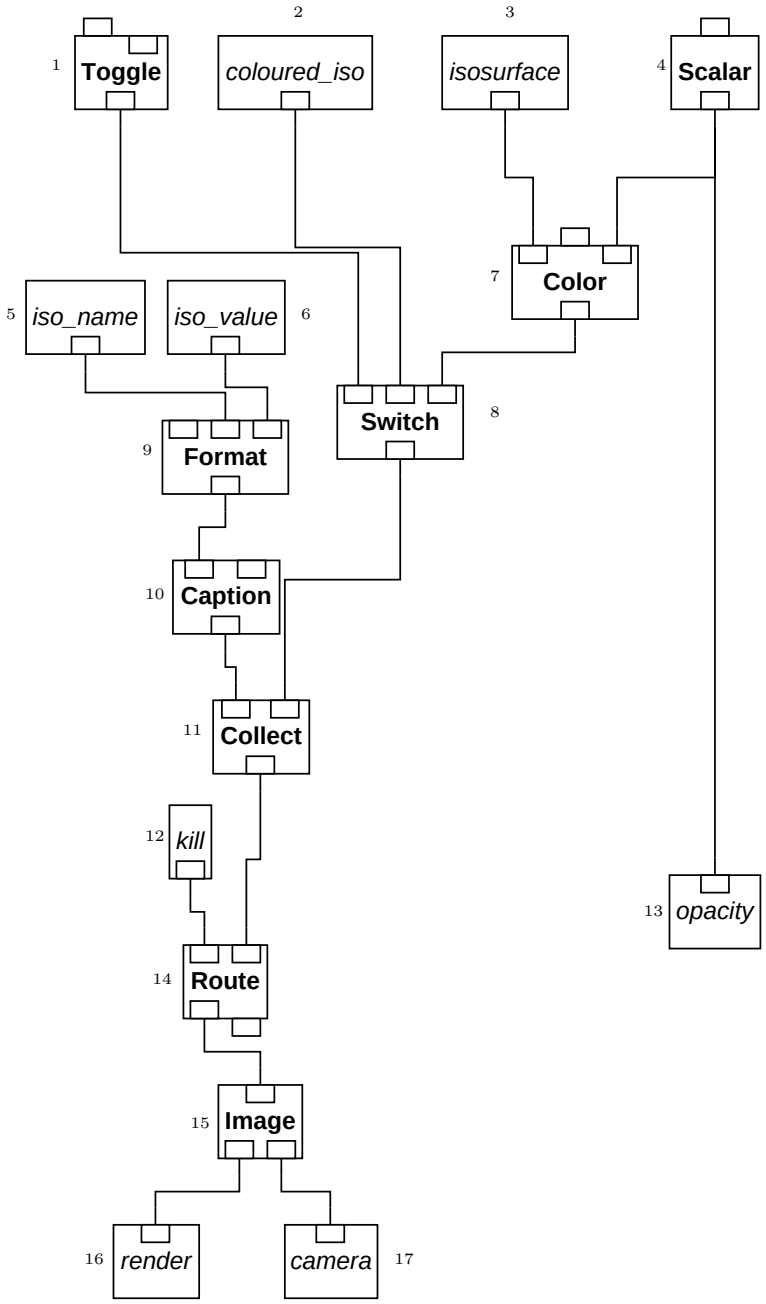
21 – `ColorBar`: Creates a vertical coloured bar annotated with associated data values, and labelled with the name of the colour field passed on by module 12 (`Attribute`).

22 – `System`: Executes the blank string passed on by module 19 (`Format`) as an operating system command. A blank command does nothing of course, so you may wonder why this is done. In fact it is a trick which can be used to ensure that certain modules get executed even if their outputs are not required. DX assumes that all operating system commands executed using a `System` module, may potentially have an effect on the final visualisation, and therefore `System` modules are always executed if possible. This means that all modules involved in creating the input command string are also executed. In the current case, this includes the two `Scalar` modules 13 and 14 which are used to obtain the data limits for colouring the iso-surface. These modules only allow the user to enter values which are in the range of their input data, *i.e.* the data passed on by module 10 (`Switch`). These limits are re-calculated every time the modules are executed. Thus, the inclusion of the `System` module (22) ensures that these limits are always re-calculated, and are thus always consistent with the data selected using module 3 (`Selector`). If this were not done, the user would be free to enter inappropriate values if the colouring field had been changed while displaying an un-coloured iso-surface.

23 – `Collect`: Collects the coloured iso-surface and the colour bar into a single object which can be displayed.

24 – *coloured_iso*: Transmits the annotated coloured iso-surface on to the rest of the network.

Demonstration “iso” - section 5



Demonstration "iso" - section 5

Purpose:

Add a caption and display the iso-surface

Description:

This section selects whether to display the iso-surface with variable or constant colour, adds a caption to the display, and displays it. It also sets the opacity of the surface.

Modules:

1 – `Toggle`: Returns 1 if the user has requested that the iso-surface should be coloured using a second field, and 2 if the iso-surface is to have a constant colour. This is determined by the state of a button on the "Surface colouring..." control panel.

2 – *coloured_iso*: Receives the coloured iso-surface transmitted by section 4.

3 – *isosurface*: Receives the original iso-surface transmitted by section 3, which has a constant blue-grey colour.

4 – `Scalar`: Obtains the opacity for the iso-surface. Zero produces a completely clear surface, and 1.0 produces a completely opaque surface. This is determined by the "Surface Opacity" widget on the "Surface colouring..." control panel.

5 – *iso_name*: Receives the name of the field used to define the shape of the iso-surface, transmitted by section 3.

6 – *iso_value*: Receives the data value used to define the shape of the iso-surface, transmitted by section 6.

7 – `Color`: Assigns the given opacity to the iso-surface. The original colour (blue-grey) is unchanged.

8 – `Switch`: Passes on the iso-surface with the colouring determined by module 1 (`Toggle`), either variable or constant. Modules connected to the un-required input are not executed.

9 – `Format`: Formats a string identifying the quantity determining the iso-surface shape, and its value.

10 – `Caption`: Creates a graphical caption containing the string created by module 9 (`Format`). This caption is "fixed" to the screen (i.e. it does not move as the viewing position is moved or rotated).

11 – `Collect`: The caption and the iso-surface are collected into a single object. Note, this module will execute even if the modules creating its inputs have not been executed. This happens if the user attempts to execute the network without first specifying an input file (see modules 4 and 7 in section 1).

12 – *kill*: Receives a flag from section 1 indicating if the execution of the rest of the network should be prevented, because of the user not giving an input file.

13 – *opacity*: Transmits the requested opacity to section 4.

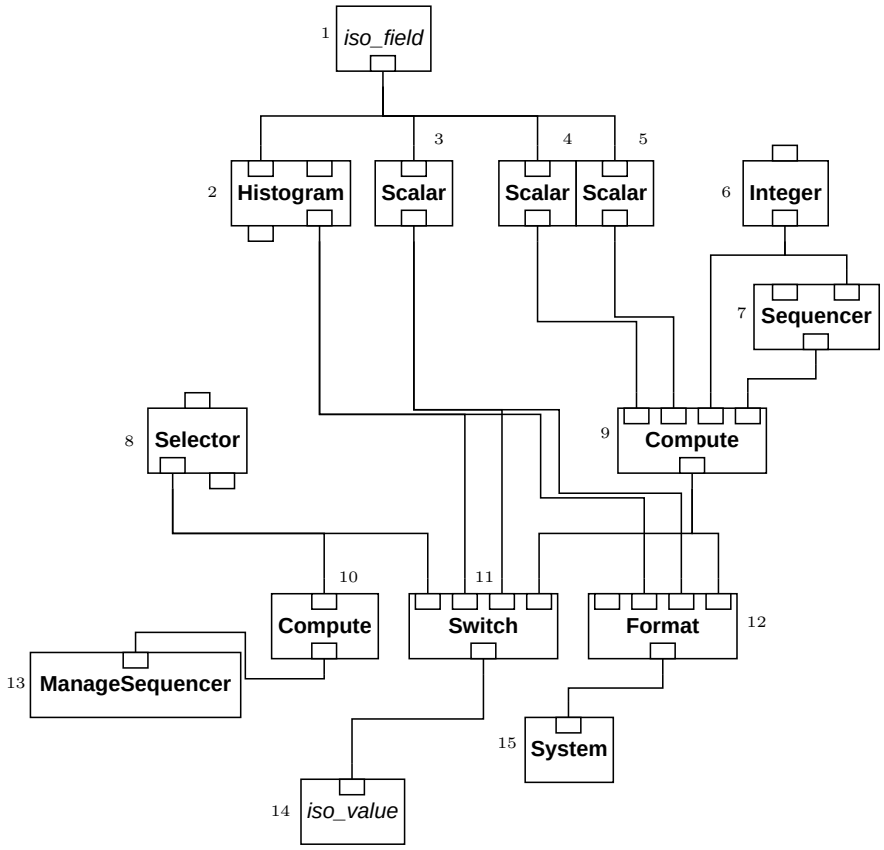
14 – `Route`: Execution is routed to module 15 (`Image`) and subsequent modules so long as the user has given an input file. Otherwise, none of the remaining modules are executed.

15 – `Image`: The iso-surface is displayed (at long last!).

16 – *render*: The final renderable object is transmitted to section 7, which is responsible for creating movies.

17 – *camera*: The current camera structure is transmitted to section 7. The camera structure contains details of how the object is projected onto the screen (viewing position and direction, screen resolution, etc). It reflects the use of the "View Control" options item in the `Image` window.

Demonstration "iso" - section 6



Demonstration "iso" - section 6

Purpose:

Obtain the data value for the iso-surface

Description:

This section determines the data value to define the iso-surface. It can either be the median of the data, a user-supplied value, or an element from a sequence of values.

Modules:

1 – *iso_field*: Receives the field data used to create the iso-surface, transmitted by section 3. This could potentially be the logarithm of the data imported from the input file.

2 – `Histogram`: Calculates the median of the data values in the field used to create the iso-surface.

3 – `Scalar`: Obtains the user-supplied data value which is to be used to defined the iso-surface. This is entered in the "Set iso-surface values..." control panel. The supplied value is constrained to lie within the data range of the specified field (as received by module 1).

4 – `Scalar`: Obtains the starting data value if a sequence of iso-surfaces are to be displayed. This is constrained to lie within the data range of the input field. It is entered in the "Set iso-surface values..." control panel.

5 – `Scalar`: Obtains the finishing data value if a sequence of iso-surfaces are to be displayed. This is constrained to lie within the data range of the input field. It is entered in the "Set iso-surface values..." control panel.

6 – `Integer`: Obtains the number of iso-surfaces to display in a sequence. The data values for each are evenly spread between the values obtained by modules 4 and 5 (`Scalar`).

7 – `Sequencer`: The sequencer causes the entire network to be repeatedly re-executed (not just this section). A new iso-surface is displayed each time. The number of re-executions is given by module 6 (`Integer`). The output is an integer index identifying the current execution. The sequence of re-executions is initiated by pressing the "Play" button in the Sequencer control panel.

8 – `Selector`: Obtains an integer identifying the method to use to select the iso-surface data value, as specified by the user using the "Value selection method" button on the "Set Iso-surface values..." control panel. A value of 1 is returned if the default (median) value is to be used as found by module 2 (`Histogram`). A value of 2 is returned if the user-supplied value obtained by module 3 (`Scalar`) is to be used. A value of 3 is returned if the sequence of iso-surfaces defined by modules 4, 5 and 6 is to be displayed.

9 – `Compute`: Calculates the data value for the current iso-surface in the sequence.

10 – `Compute`: Outputs 1 if the value obtained by module 8 (`Selector`) is 3 (i.e. a sequence of iso-surfaces is to be displayed), or zero otherwise.

11 – `Switch`: Selects the current iso-surface value on the basis of the value returned by module 8 (`Selector`).

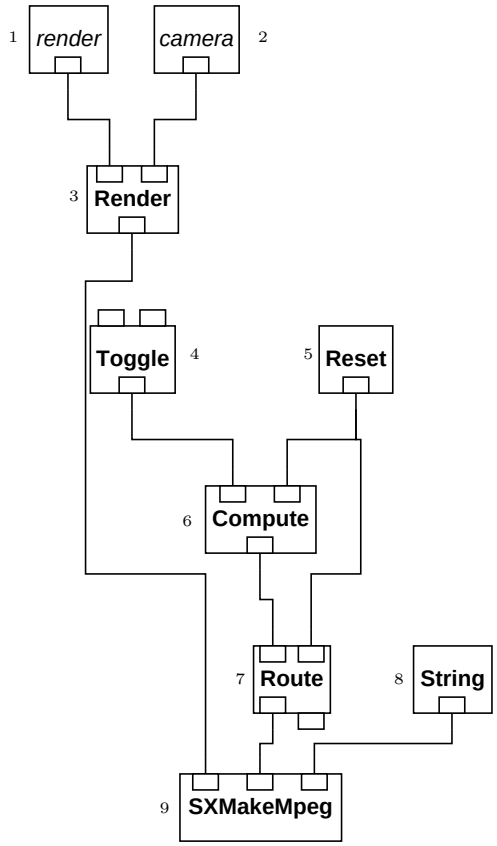
12 – `Format`: Formats a blank string. Note, the blank string is permanently assigned to the left hand input tab using the module's configuration dialogue box. The other input tabs are only connected in order to trigger execution of the module. The values they provide are not actually used.

13 – `ManageSequencer`: Ensures that the sequencer control panel is visible if the user wants to display a sequence of iso-surfaces, and closes it otherwise. Note, the user could do this manually by selecting the "Sequencer" item from the "Execute" menu. Including this module removes the need for user-intervention.

14 – *iso_value*: Transmits the data value required for the current iso-surface.

15 – `System`: Executes the blank string passed on by module 12 (`Format`) as an operating system command. This is done to ensure that modules 2, 3, 4, 5 and 6 are always executed and thus kept consistent with any change in input data. This is necessary because module 11 (`Switch`) would otherwise suppress the execution of some of these modules. See module 22 in section 4 for further discussion of this "trick".

Demonstration “iso” - section 7



Demonstration "iso" - section 7

Purpose:

Gather the frames for a movie and encode them into an MPEG file.

Description:

The displayed image is re-created incorporating any axes, rotation, shift, etc, introduced by the use of the `Image` window controls. This image is saved on disk as a "PPM" file by the `SXMakeMpeg` macro if the user has enabled the saving of images. When all images have been saved, they are encoded into an MPEG file using the Berkeley "`mpeg_encode`" software, which should be installed on the host system.

Modules:

1 – *render*: Receives the displayed object, transmitted by section 5.

2 – *camera*: Receives information describing the position from which the displayed object is viewed, transmitted by section 5.

3 – `Render`: Projects the object received by module 1 (*render*) into a 2-d image, viewed in the way described by the camera received by module 2 (*camera*). This image will be a copy of the image displayed in the `Image` window.

4 – `Toggle`: Obtains the setting of the "Save frames" button on the "MPEG control" control panel. It returns 1 if the button is pressed (i.e. if the current displayed image is to be saved so that it can become the next frame in an MPEG movie), or zero otherwise.

5 – `Reset`: Obtains the setting of the "Create MPEG" button on the "MPEG control" control panel. It returns 1 if the button is pressed (i.e. if the current collection of saved frames is to be encoded into an MPEG file), or zero otherwise. After execution of the module, the button is automatically reset to its "released" position.

6 – `Compute`: Returns 1 if either the "Save frames" or the "Create MPEG" button is pressed, and zero if neither of them are pressed.

7 – `Route`: Causes execution to be routed to module 9 (`SXMakeMpeg`) only if one of the two buttons have been pressed, i.e. if module 6 (`Compute`) outputs 1. Otherwise, module 9 is not executed.

8 – `String`: Obtains the name of the file in which to store the encoded MPEG movie. This is supplied by the user on the "MPEG control" control panel.

9 – `SXMakeMpeg`: This either saves the current image as a "PPM" file on disk, or encodes the current collection of PPM files into an MPEG movie. It is a macro, and expects the Berkeley MPEG encoder ("`mpeg_encode`") to be available on the host system.

13 Acknowledgements

Ewan Brown explained the intricacies of the various Unix commands for displaying the amount of swap space available. Alan Chipperfield advised on the configuration and operation of the Starlink CONVERT package. Mike Lawden made several useful comments on a draft version of the document.

References

[1] *IBM Visualization Data Explorer: QuickStart Guide*, Second Edition, 1995, document reference number: SC34-3262-01.

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

[3] *IBM Visualization Data Explorer: User's Reference*, Third Edition, September 1995, document reference number: SC38-0486-02.

[4] *IBM Visualization Data Explorer: Programmer's Reference*, Sixth Edition, 1995, document reference number: SC38-0497-5.

[5] SC/2.3 *The DX Cookbook* by A.C. Davenhall, 1 October 1997 (Starlink).

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

[7] SUN/33.4 *NDF Routines for Accessing the Extensible N-Dimensional Data Format* by R.F. Warren-Smith, 21 March 1995 (Starlink).

[8] SUN/55.9 *CONVERT A Format-conversion Package* by M.J. Currie, G.J. Privett and A.J. Chipperfield, 7 December 1995 (Starlink).

[9] SUN/95.9 *KAPPA — Kernel Application Package* by M.J. Currie, 9 December 1995 (Starlink).

A The ndf2dx Conversion Utility

This appendix describes `ndf2dx` in detail.

NDF2DX Convert a Starlink NDF structure into a DX native data file NDF2DX

Description: NDF2DX converts a 1, 2 or 3 dimensional Starlink NDF structure to a native format DX data file containing a single field with the following array components:

`data` – This is copied from the DATA component of the NDF. It is given a "dep" attribute indicating whether the values in it are in one-to-one correspondence with the `positions` component, or the `connections` component (see parameter DEP). Bad values are replaced by zero, and identified in the `invalid positions` or `invalid connections` components. The created component is scalar and of type `int`, `float` or `double`, depending on the data type of the NDF.

`positions` – This is derived from the AXIS component of the NDF and is stored as a product of regular arrays if possible. Default values are used if the AXIS structure is undefined, in which case each pixel centre is given the corresponding pixel co-ordinates. If the user indicates that the data are positions-dependent (see parameter DEP), then the `positions` component holds the co-ordinates of the NDF pixel centres. If the data are connections-dependent, then the `positions` component holds the co-ordinates of the corners of the NDF pixels. All positions are stored as `float` values.

`connections` – This component identifies adjacent positions which can be connected together to form cubic cells. If the data are connections-dependent then these cells correspond to NDF pixels. Otherwise they form a grid in which each cell corner occurs at an NDF pixel centre.

`variance` – This is copied from the VARIANCE component of the NDF. It is given the same "dep" attribute as the `data` component. It is not created if the VARIANCE component is undefined, or if the PVAR parameter is set false. Bad values are replaced by zero and marked in the appropriate `invalid ...` component.

`invalid positions` – This is a list of indices into the `positions` component, identifying positions which have bad DATA or VARIANCE values. It is of type `integer`. It is not created if there are no bad values or if parameter PBAD is given the value FALSE or if the data are connections dependent (see parameter DEP).

`invalid connections` – This is a list of indices into the `connections` component, identifying cells which have bad DATA or VARIANCE values. It is of type `integer`. It is not created if there are no bad values or if parameter PBAD is given the value FALSE or if the data are positions dependent (see parameter DEP).

In addition, the field is given the following attributes:

`name` – This is a string object which is set equal to the TITLE component of the NDF (or "<undefined>") if there is no TITLE component.

`axis labels` – This is a string list containing one string for each axis. Each string consists of the corresponding NDF AXIS LABEL component followed by the axis UNITS in parentheses. This attribute is used by the DX `AutoAxes` module.

`ndf_label` – A string attribute holding the NDF LABEL component, or "<undefined>" if there is no LABEL component.

`ndf_units` – A string attribute holding the NDF UNITS component, or "<undefined>" if there is no UNITS component.

Usage:

```
$SX_DEV/ndf2dx in out bmode dep pvar pbad axes
```

Parameters:

IN = NDF (Read)
The input NDF.

OUT = FILENAME (Write)
The output file, including "`.dx`" suffix if required. Existing files with the given name will be over-written.

BMODE = LOGICAL (Read)
If YES, then the output file will hold data arrays in binary form, otherwise they will be in text form. [YES]

DEP = LITERAL (Read)
The data dependency. This can be either "connections" or "positions" (case insensitive, unambiguous abbreviations are allowed) Positions-dependent data values are considered to be instantaneous samples from a smoothly varying field, taken at the NDF pixel centres. Connections-dependent data values are considered to be values which describe the whole pixel volume (for instance, the data values may be connection dependent if they represent the integral of a field over the pixel volume). DX interpolates smoothly in a position-dependent data array to find data values at un-tabulated positions, but uses the value of the cell in which the position lies if the data are connection dependent. See the DX *User's Guide* Section 2.1 for more information. [POSITIONS]

PVAR = LOGICAL (Read)
If TRUE then any VARIANCE component in the input NDF will be copied to a `variance` component in the output DX field. [YES]

PBAD = LOGICAL (Read)
If TRUE then an `invalid connections` or `invalid positions` component will be added to the output field identifying any bad DATA or VARIANCE values in the input NDF. [YES]

AXES = LOGICAL (Read)
If TRUE then any axes components in the NDF are copied to the output file. If FALSE then any axes components are not copied and the output file will have the default units of pixels. [YES]

Examples:

```
ndf2dx jet jet.dx
```

This example converts the NDF in file `jet.sdf` into a native DX structure in file `jet.dx`.

Implementation Status:

- This routine cannot currently process data values which are of type COMPLEX.

B Obtaining a Copy of DX

DX is a commercial product produced and sold by IBM. There is no CHEST agreement for it, though there is an educational discount. As a guide the following prices applied at the time of writing (November 1995):

	Full List Price	Educational Price
Node locked licence	£4700	£940
Floating licence	£5895	£1179

These prices are exclusive of VAT. A node-locked licence allows any number of concurrent invocations of DX on a single workstation (though most Starlink machines probably could not cope with multiple invocations). A floating licence allows only a single invocation at any instant, but it may be on any machine at your site. The licences permit you to run the version that you have bought in perpetuity. However to obtain the next version you would need to buy an upgrade, which costs about £200.

Normally sites will obtain a copy by including an item for DX in their annual bid submitted to Starlink. If the bid is successful a copy of DX will be bought. Alternatively, of course, a site could simply purchase a copy using local funds. In either case the person at Starlink to contact in the first instance is C.A. Clayton (e-mail `cac@star.rl.ac.uk`) who will supply the necessary details.