Final Report on NASA/AISRP Proposal Order No. S-46494-E, An Interactive Interface for NCAR Graphics.

BILL BUZBEE, BOB LACKMAN, and ETHAN ALPERT, NCAR

NASA Funding

Funding of this proposal occurred between July 1991 and July 1994. During the lifetime of this project, quarterly and tri-annual reports were issued. In 1992 and 1993 progress report presentations were given at the annual summer AISRP Workshop in Boulder, CO. Since the full project "An Interactive Interface for NCAR Graphics" was much larger in scope than covered by the size of the NASA grant, the statement of work to be covered by the grant was defined on March 11, 1991 to be a reasonable subset of the total effort. This report, however, will discuss the entire project, since that will have more meaning to the reader.

NCAR Interactive

The goal of the NCAR Interactive project was to create, from existing utilities, a fully integrated scientific visualization environment. To be fully integrated, this environment had to provide a means of reading and writing data, a means of manipulating that data, and a means of visually analyzing the data interactively. It had to be easy to use, work efficiently, and be user-extensible. Furthermore, this environment had to support users with a wide range of skill levels: from users with extensive programming and technical skills to users who have not developed these skills. To respond to these varied needs, we created a set of interfaces which allow for skill levels from the occasional user or novice to the expert user who can go so far as to extend the software.

Integrating interactive data access and manipulation with the visual analysis techniques of NCAR Graphics into one system was deemed to have tremendous benefits to current and future users of NCAR Graphics. This saves users from writing custom data-reading code as a prerequisite for producing graphics. It simplifies and accelerates the process of tailoring a data plot for publication. Users can develop customized plot specifications that can be saved and reused. New users have multiple ways to learn to use NCAR Graphics. Users who develop applications with NCAR Graphics will find the new API to be consistent across utilities and supportive of many interactive demands.

Three major components of the NCAR Interactive system accomplish these goals: a toolkit programming interface library, a text-based data manipulation and graph specification language interpreter, and a "point-and-click" graphical user interface. All of these interfaces accomplish the same result from the user's point of view, a user-specified display of data.

Functional Specification

You can acquire a PostScript copy of the functional specification of the NCAR interactive project via anonymous ftp from ftp.ucar.edu. The path to the file is ncari/fspec.tar.Z. A copy of this document is attached to this report for your convenience. Any specific questions with respect to Version 4.0 can be sent to ncargproto@ncar.ucar.edu.

Figure 1 shows the various components of the NCAR Interactive system. These components are briefly described below. For a detailed description of the NCAR Interactive system, please refer to the Functional Specification document.

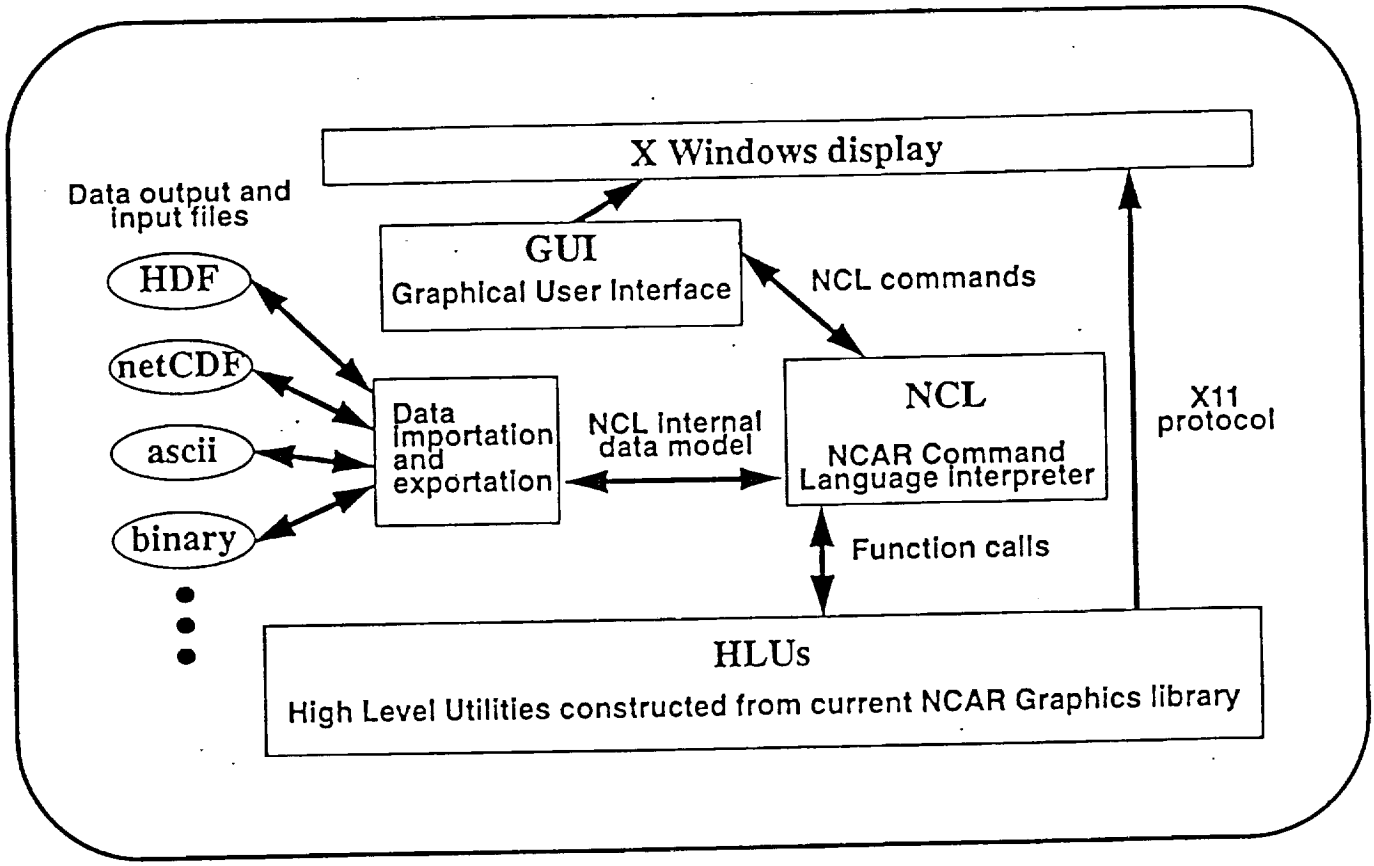


Figure 1. NCAR Interactive Functional Diagram

LLU Description

The existing Version 3.2 of NCAR Graphics, a set of about 500 low level graphics routines with Fortran and C bindings, will be known as the Low Level Utilities (LLUs) in Version 4.0. The current Version 3.2 Fundamentals manual, which tells how to use this library, will be relabeled "Low Level Utility Fundamentals." This library will continue to be supported to ensure that the many current NCAR Graphics codes spread throughout the research community will continue to run.

However, when users generate new codes they will be urged to try one of the new interfaces: the High Level Utilities (HLUs), NCAR Command Language (NCL), or Graphical User Interface (GUI).

HLU Description

The toolkit programming interface library builds upon the existing NCAR Graphics low level libraries. The new toolkit library is called the High Level Utilities (HLUs). The HLUs represent a new interface to the NCAR Graphics

library with a great deal of the underlying low-level implementation details hidden from the application programmer.

The main purpose is to extend NCAR Graphics to support the demands of interactive graphics, primarily giving users the ability to change features of the output frame efficiently and quickly. Other features, like being able to retrieve information about the current state of an HLU and retrieving data coordinate information, are built into the HLUs.

Other purposes of the new programming interface to NCAR Graphics are to introduce a new level of interface consistency between utilities and to consolidate most of the existing functionality into a new set of high level utilities that produce the same or better results, but reduce the overall complexity of using the package. This will make learning NCAR Graphics at the programming level much easier. The toolkit has entry points from both FORTRAN and C.

The model chosen to implement the HLUs is essentially a simplified version of the X-Window System Toolkit widget. Many changes have been made to the model, but the main concepts are still valid. This model supports the demands of an interactive environment in which different plots are created and displayed at different times with different data and configurations. The X-Window System Toolkit widget model encapsulates the set of functions that draw a given class of plot with a set of data needed to configure it. These data can be line widths, fill colors, transformation styles, actual data, etc. As is the case for X-Window System widgets, multiple instances of each type of plot object can be created, destroyed, drawn, and reconfigured without affecting any of the other instances. This model also supports derived classes of HLU objects (inheritance) and mixing of separate classes to form new classes.

HLU objects are configured just like X-Window System widgets, with respect to resources. A typical example of an HLU resource name is "NhlTxFont", which sets the font for a TextItem object. Only five functions are needed to use the HLU library: NhlCreate to instantiate an object; NhlDraw to draw it; NhlGetValues to determine the current configuration of the object, NhlSetValues to reconfigure it; and NhlDestroy to get rid of it and free any memory it uses. A unique integer ID is returned by the NhlCreate call for each instance of an object. This ID number is used to identify the object in the remaining functions.

Additional features of an HLU object are its ability to maintain its own transformation and its ability to map screen positions into data coordinates. Each object also knows how much area on the screen it uses and can report its bounding box in frame coordinates. Thus, when objects change size they can automatically scale text and line lengths proportionally. All of these features are necessary for interactive operations.

HLU Objects

At release 4.0, the HLU library will contain the following objects:

TextItem	Generic text placement object, supports filled fonts
LabelBars	Generic filled label bar, used by Contour object
Legends	Generic legend that displays both symbols and lines with text, used by XyPlot object
Titles	Positions titles around plots
TickMarks	Draws tick marks around plots in 5 transformation styles (LOG, LINEAR, IRREGULAR, GEOGRAPHIC, and TIME)
XyPlot	Draws curves and symbols, uses Titles, TickMarks, and Legends
Contour	Draws filled contours with and

	without maps, automatically generates a LabelBar and TickMarks
MapPlot	Generates maps in any of the 10 standard NCAR Graphics transformations
Workstation	Allows output to be directed to workstations of type NCGM, PostScript, or an X11 window

To be added at release 4.1 are:

Histogram	Creates a histogram plot, uses TickMark object
StreamLine	Creates a streamline with and without maps
Vectors	Creates a vector with and without maps
3D	A set of 3D HLUs (iso-surfaces, surfaces...)

NCL Description

The heart of the system is the data manipulation and graph specification language, called the NCAR Command Language (NCL). NCL provides easy and intuitive access to datasets and allows users to explore and process their data prior to visualization. Since datasets often come in a variety of data formats, grid sizes, grid resolutions, and units, very different datasets often need to be combined, compared, and used at the same time. Currently, specialized applications must be developed to read individual datasets and transform them into a form that is compatible with other datasets being used, as well as with the graphics package being used.

NCL allows different datasets, in different storage formats, to be imported into one uniform and consistent data manipulation environment. The primary data format used internally by NCL is the netCDF data format. NCL doesn't place any restrictions or conventions on the organization of input netCDF files.

NCL is a complete programming language that provides flexibility and configurability. In NCL, the primary data type is the data file record. A data file record stores one or more variables, dimension information, coordinate variable information, and attribute information as one NCL object. A binary file can be input, then dimension names, variable names, attributes and coordinate variables can be assigned to it using NCL language constructs. The resulting file record can be written to any of the currently supported formats, including netCDF, without writing a single line of source code.

NCL also provides the ability to create and manipulate the HLU graphical objects utilizing the same resources available through the HLU toolkit interface.

NCL's function set contains built-in data processing and mathematical functions. Users can extend NCL functions to implement custom data processing techniques and custom data ingestion.

NCL commands can be interactively interpreted and executed in line mode. In addition, an API for NCL is under development which provides GUI writers a uniform way to do data access, data manipulation, and visualizations from within their application using NCL scripts. This simplifies the implementation of application-specific GUIs by reducing the amount of design time spent on the visualization code and the data access code.

Data Import/Export

Figure 2 shows an NCL script for the import of a netCDF dataset. Note that variables can be extracted using latitude and longitude ranges rather than numeric indices.

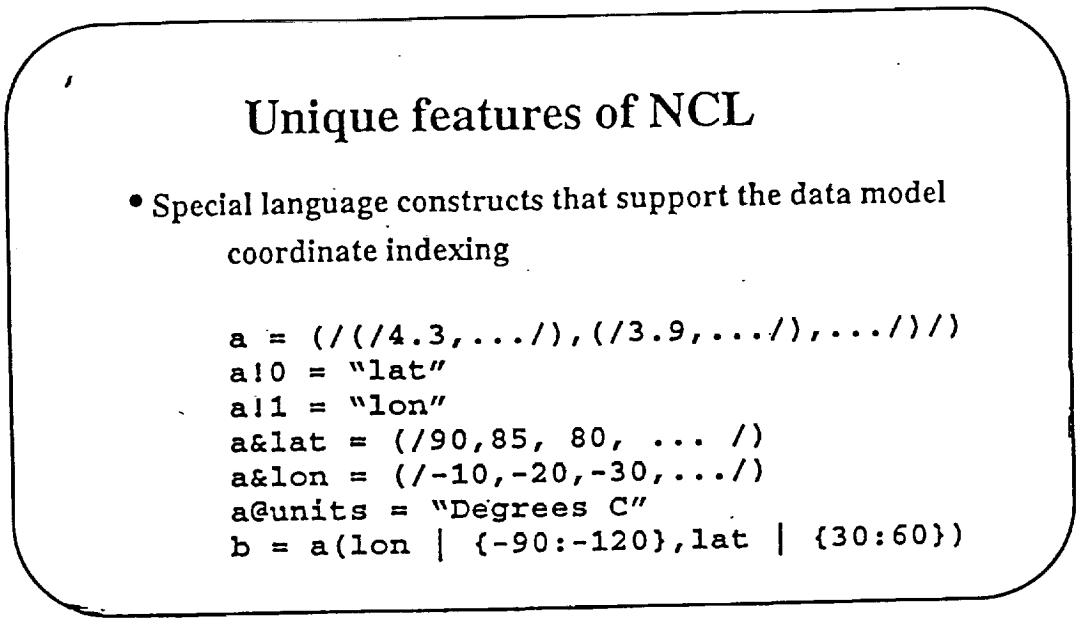


Figure 2. NCL script for array access using coordinates

Visualization Blocks

Figure 3 shows several visualizations created by an NCL script. NCL has a visualization block feature that allows you to easily set the resources of the HLU plot objects. Figure 3 shows a typical visualization block.

GUI Description

The GUI for NCAR Interactive is built on top of NCL, which is built on top of the HLUs. Thus the GUI will provide access to the data access and manipulation tools as well as the visualization specification. The GUI simplifies many of the user's tasks by providing a "point-and-click" style interface for selecting data, adding data to a plot, and positioning plots within the output frame.

Furthermore, the GUI allows users to "probe" data visualizations by attaching NCL scripts that perform some analytical function to various mouse and key press events, making true data interaction possible.

Figure 4 shows a page layout on which a number of graphical objects have been drawn including three instances of the TextItem object. Each instance can be selected, resized, or have its resources altered.

At Version 4.1, the GUI will also provide editors for color maps, fill patterns, line styles, area masking, map projection specification, and animation in addition to selections of all the available resources.

Hypertext Documentation

All of the Version 4.0 interfaces are being documented on-line in hypertext for search and display access. Figure 5 shows the NCAR Graphics documentation home page accessible via the URL http://cyclone:7777/nggenrl/ctrldisplay.html.

In the event that this URL address changes between the writing of this report and your attempt to access it, please send e-mail to ncargfx@ncar.ucar.edu requesting the new NCAR Graphics home page URL.

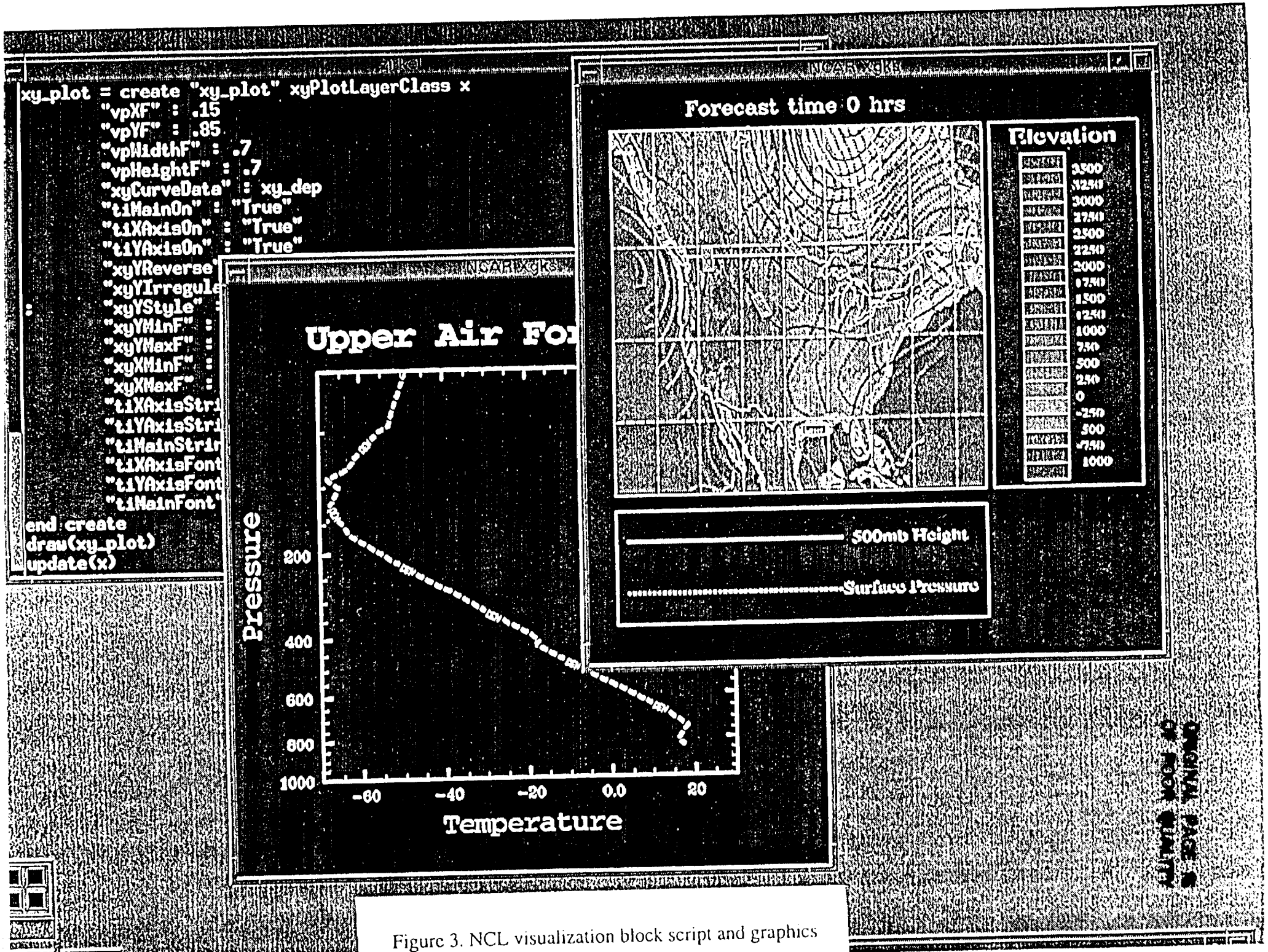


Figure 3. NCL visualization block script and graphics

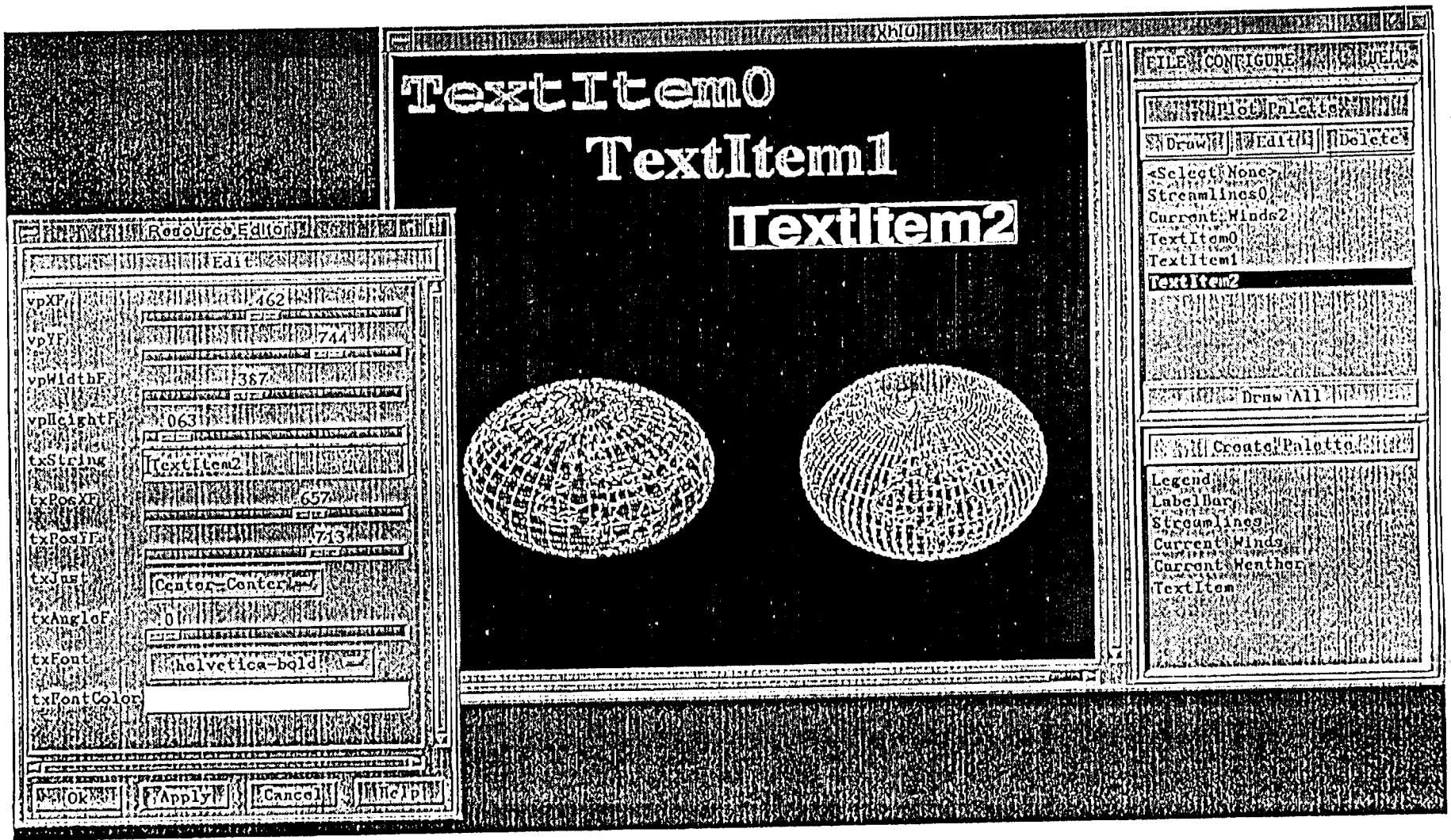


Figure 4. GUI manipulation of HLU plot objects

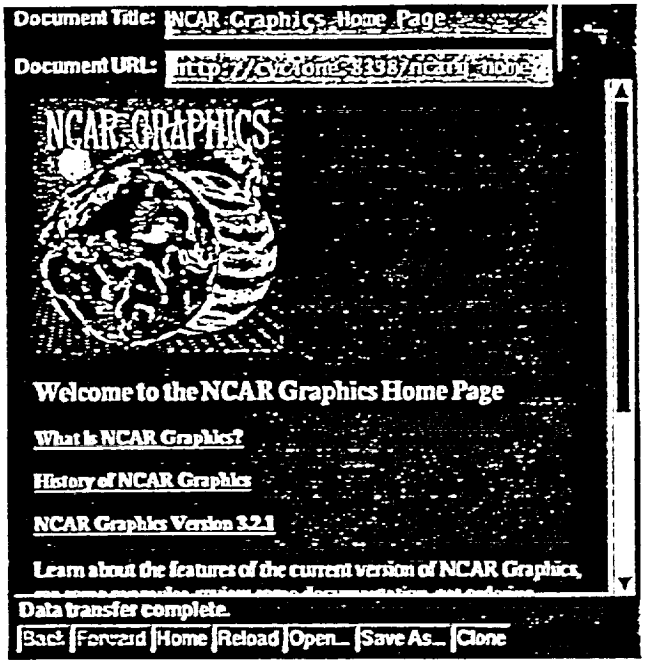


Figure 5. NCAR Graphics documentation home page

The Version 4.0 documentation set under development includes:

- User Guide
  The User Guide describes the various package interfaces and how to use them.

- Quick Start Guide
  The Quick Start Guide is a series of examples that facilitate self training.

- Reference Manual
  The Reference Manual describes how the software works. It specifies all NCL functions, their syntax, and the actions they produce. It also specifies all HLU resources (option switches) and how they are set.

- Low Level Utility Manuals.
  The low level library manuals, called "NCAR Graphics Fundamentals", "NCAR Graphics Contouring and Mapping Tutorial", and "NCAR GKS-0A User Guide" at release 3.2 of NCAR Graphics have also been converted to hypertext. Figure 6 shows a sample page from the Fundamentals manual.

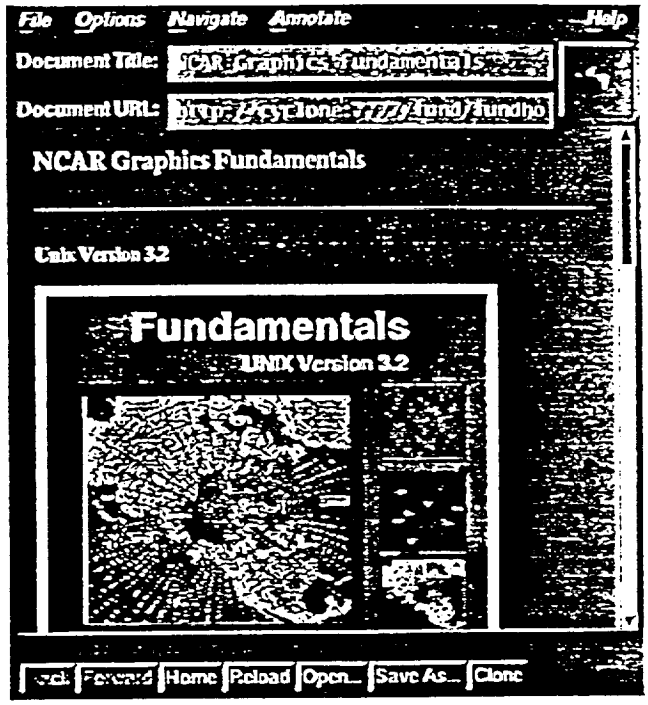


Figure 6. Fundamentals Document available in Mosaic

Input/Output

Currently implemented input formats are netCDF, ASCII, and binary. The next set of formats to be implemented will be HDF and GRIB.

Output can be to an X11 window, an NCGM file, or a PostScript file. (The private encoding NCAR CGM format can be replaced by the CGM format through the selection of an environment variable.) All variations of PostScript are supported including mono and color for EPS, EPSI, and standard PostScript. Output can be in landscape or portrait mode.

Fonts and Symbols

A high quality set of fonts are available with numerous font options as shown in Figure 7. Since the fonts are stroked, characters can be written at any size or orientation angle. Available fonts include the math symbols and the full set of standard WMO meteorological symbols.

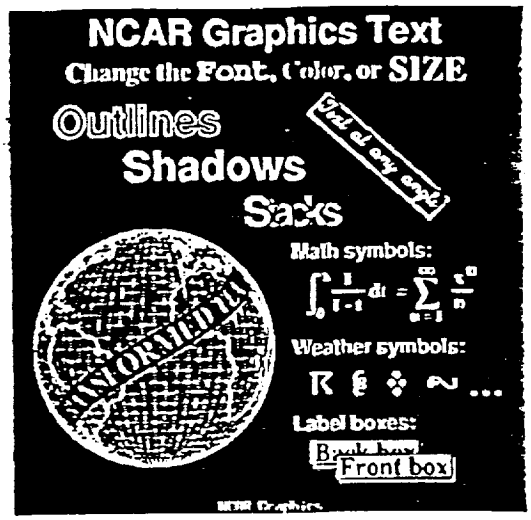


Figure 7. Text options available in NCAR Graphics

In addition, a set of symbols for standard weather have been created as shown in Figure 8.

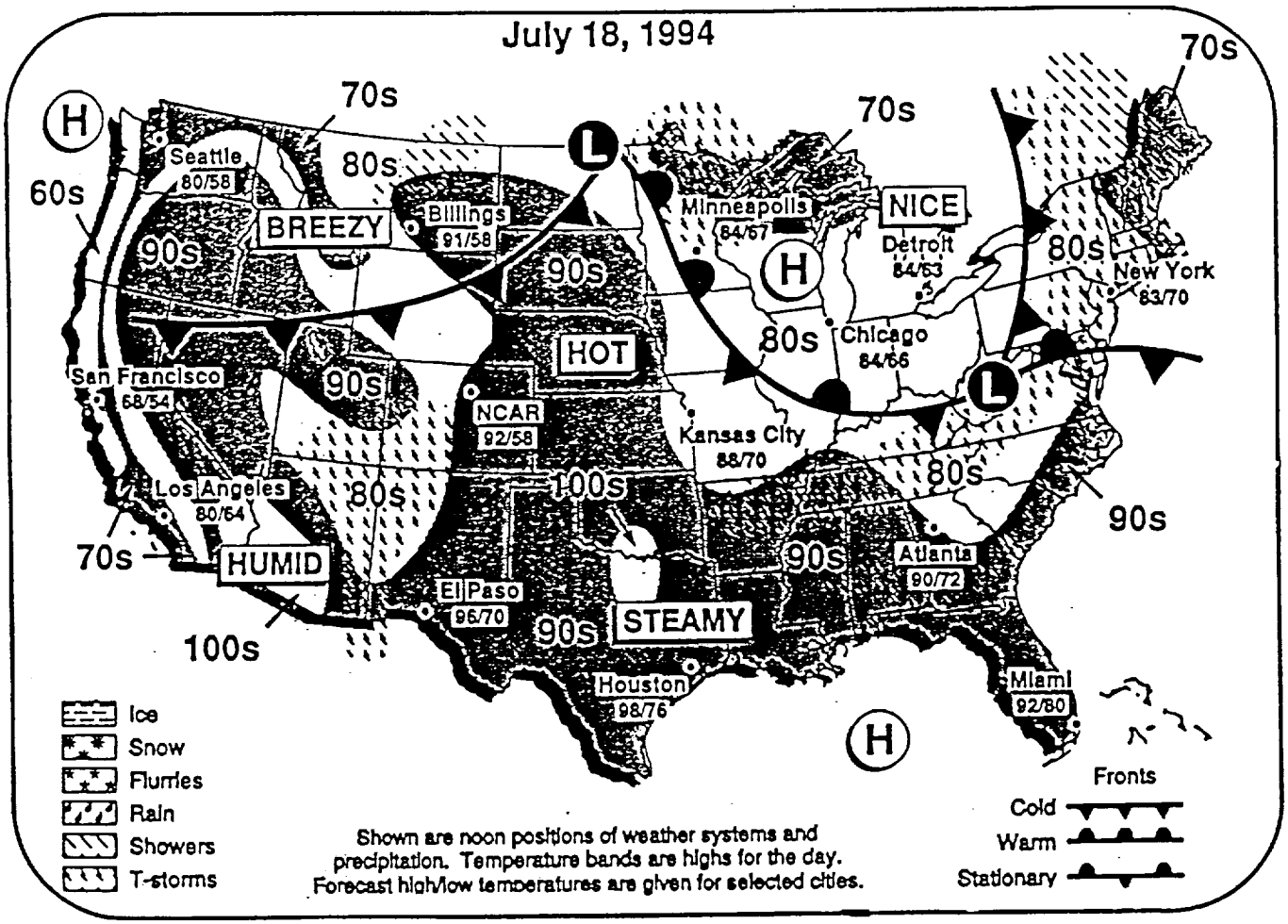


Figure 8. Creating weather maps with NCAR Graphics special symbols

System Requirements

NCAR Interactive is intended for use on all UNIX platforms that are popular in universities and other research communities. This includes Suns, SGIs, HPs, IBMs, Digital Alphas, and DECstations. It is also intended for installation on Cray UNICOS systems; however, certain interactive features may not be as useful on heavily loaded Crays. The first release of Version 4.0 may not support all of these systems: Suns and SGIs will be done first. Porting to the other systems will then follow.

Conclusions, Recommendations, and Acknowledgments

NASA funding was instrumental in helping us to complete the first release of an interactive version of NCAR Graphics in a reasonable time frame. NCAR Graphics is a research community resource whose value will be significantly enhanced by the new interactive interfaces. At the time of this writing, NCAR Graphics Version 4.0 is entering the test phase. Release of the software is expected in early 1995.

Version 4.0 will contain a limited but useful set of functionality; however, successive releases of the software are planned which will continue to expand the functionality of the High Level Utilities, the NCAR Command Language, and the Graphical User Interface. Moreover, HLU objects, NCL functions, and GUI functionality are user-extensible so they can grow in whatever direction is necessary to meet the changing needs of future researchers.

Although the creation of an interactive interface for NCAR Graphics was not a basic research project, it was a project that will significantly improve the productivity of the broad research community. It is our recommendation that more projects of this nature should be included in future agency grants. Basic research into new technological areas is very important, but it is also important that those researchers have a set of high quality, extensible, easy-to-use tools to facilitate their research.

NCAR INTERACTIVE

FUNCTIONAL SPECIFICATION

DRAFT

ALPERT, BOOTE and SCHEITLIN

SECTION 1 NCAR Interactive overview

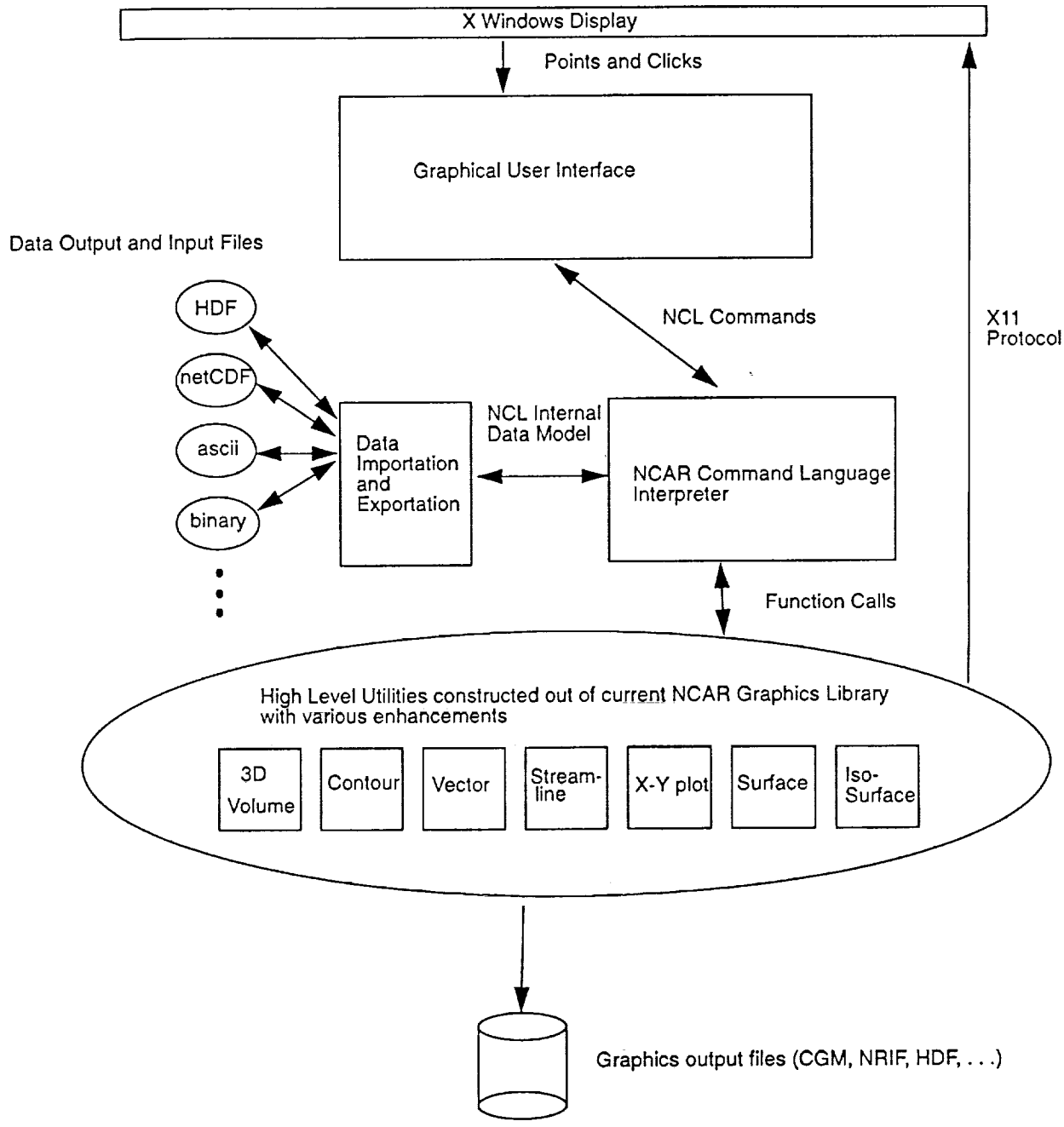
1.1 Introduction

The goal of the NCAR Interactive project is to create, from existing utilities, a fully integrated scientific visualization environment. To be fully integrated, this environment must provide a means of reading and writing data, a means of manipulating that data, and a means of visually analyzing the data interactively. It must be easy to use, work efficiently, and be user-extensible. Furthermore, this environment must support users with a wide range of skill levels: from users with extensive programming and technical skills to users who have not developed these skills.

Integrating interactive data access and manipulation with the visual analysis techniques of NCAR Graphics into one system will have tremendous benefits to current and future users of NCAR Graphics. In many cases, users will no longer need to write custom data-reading code as a prerequisite for producing graphics. The process of tailoring a data plot for a publication will be greatly simplified and accelerated. Users will be able to develop customized plot specifications that can be saved and reused. New users will have multiple ways in which they can learn to use NCAR Graphics. Users who develop applications with NCAR Graphics will find the new API to be consistent across utilities and supportive of many interactive demands.

Three major components of the NCAR Interactive system will accomplish these goals: a toolkit programming interface library, a text-based data manipulation and graph specification language interpreter, and a "point-and-click" graphical user interface. All of these features, from the user's point of view, accomplish the same result: a user-specified display of data. Figure 1 shows the overall system architecture.

FIGURE 1 System Structure



The toolkit programming interface library needs to be developed from the existing NCAR Graphics code. From now on, this toolkit library is called the High Level Utilities (HLUs). The HLUs will be a new interface to the NCAR Graphics library with a great deal of the underlying low-level implementation details hidden from the application programmer.

The main purpose is to extend NCAR Graphics to support the demands of interactive graphics, primarily giving users the ability to change features of the output frame efficiently and quickly. Other features, like being able to retrieve information about the current state of an HLU and retrieving data coordinate information, need to be built into the HLUs.

Other purposes of the new programming interface to NCAR Graphics are to introduce a new level of interface consistency between utilities and to consolidate most of the existing functionality into a new set of high level utilities that produce the same or better results, but reduce the overall complexity of the package. This will make learning NCAR Graphics at the programming level much easier. The toolkit should have entry points from both FORTRAN and C, but may be written in either language.

The heart of the system is the data manipulation and graph specification language, from now on called the NCAR Command Language (NCL). NCL will allow users to import data from a variety of data formats including ascii files, interactively manipulate the data, and render the data.

Interactive data manipulation includes selecting region-of-interest data, thinning data, interpolating irregular grids to regular grids, performing curve fits, removing corrupted data points, and performing useful transformations such as unit conversion. Because of resource constraints and potential oversights in exactly what type of data manipulation is needed, NCL must allow the user to incorporate customized data

manipulation and data ingestion methods into NCL in the form of user-defined functions.

Data rendering is primarily what NCAR Graphics has done all along. NCL will allow users to generate plots in various styles for 1D, 2D, and 3D data. These plot styles include but should not be limited to line, scatter, contour, histograms, isosurface, streamline, and 3D volume rendering. Capabilities for annotation and labeling are also required. Screen animation capability should be added as soon as possible.

The NCAR Interactive GUI should provide a "point-and-click" abstraction of NCL. Through the use of menu bars, buttons, scale bars, and direct manipulation, the user should be able to easily and intuitively manipulate data and specify plots. The GUI should produce NCL source code as output.

1.2 Guide to this document

This document is a communication tool connecting the programmers assigned to this project and the user community. The decisions this document seeks must be made now; they will affect NCAR Graphics for several years. These decisions concern the design of NCL, the organization and functionality of the HLUs, and the look and feel of the GUI. After the design of this application is finished and coding begins, major alterations in functionality may not be undertaken for years.

Users and programmers share responsibility for this project; they need to communicate clearly and often to complete it successfully. This document describes our current plans, and we need user input to either confirm or alter these plans as soon as possible.

Because different users have different needs and intended uses for this software, this section outlines this document so reviewers can read about the features and functionality that concern them most.

1.2.1 Software and hardware requirements

These sections outline the software libraries, operating systems, and architectures that NCAR Interactive will use. The hardware section also contains a list of the hardware output devices. The software section lists the output file formats used for storing data and graphics.

1.2.2 High Level Utilities

As mentioned, a set of new programming interfaces to the NCAR Graphics utilities is being developed to support this project. This set of interfaces is called the High Level Utilities (HLUs). It is important to distinguish between the NCAR Interactive application and the High Level Utilities programming interface. The NCAR Interactive application uses the HLUs, and the HLUs are just a library that any programmer can use to develop programs.

The HLU sections are mainly meant for programmers who may use the new interfaces to NCAR Graphics to build applications, specifically interactive applications. The features and capabilities of this new API are outlined briefly in the section titled "New programming interface to NCAR Graphics (High Level Utilities)". A more detailed description of the features provided by each of the HLUs can be found in the section titled "Detailed resource descriptions."

Since the HLUs are just a toolkit, there are no built-in data access and manipulation routines; application programmers must still read, write, and process data. However, the NCAR Interactive application will have built-in data handling capabilities.

Users who want to understand more about the graphics capabilities of the NCAR Interactive application should read the first section on the High Level Utilities.

1.2.3 NCAR Command Language

The NCAR Command Language (NCL) provides interactive data access and manipulation, a fundamental requirement for interactive visualization systems. The NCL also serves to import and export data in various formats and to configure the output graphics.

It is very important to understand how the command language fits into NCAR Interactive. There are three intended uses of the NCAR command language. First, the command language can serve as a text-based interface to NCAR Interactive when it is not possible to use an X-Window workstation or terminal. Second, scripts written in the command language can be submitted to the NCAR Interactive command language interpreter. The interpreter then executes the submitted script in a batch style mode. Finally, there are two ways the NCAR Command Language fits into the NCAR Interactive graphical user interface. The GUI can generate command language scripts, which can then be used for batch production of visualizations. In addition, scripts can be typed into the GUI, when selecting data, to process selected data or to derive new datasets from several datasets.

Because the command language is such an integral part of the NCAR Interactive application, it is recommended that all intended users of this application read this section. Programmers who are only interested in the High Level Utilities toolkit specification probably don't need to read the command language specification.

1.2.4 The Graphical User Interface

This section provides a general description of the various GUI components and their functionality. This section describes what users should be able to do using the GUI.

A general interaction and usage model is not provided in this section. This is being developed by presenting GUI prototypes and mock-up to users and responding to their feedback. Describing GUIs in a textual manner is difficult and inefficient at best. As user interface conventions are adopted, they will be added to this document. For the time being, this section serves only as a reference to what the GUI will do. It does not describe how it operates.

Users who are interested only in the GUI for NCAR Interactive should read this section first.

SECTION 2

SOFTWARE REQUIREMENTS

NCAR Interactive must use the utilities defined in the NCAR Graphics 4.0 distribution plus the PolyPaint 3-D rendering source.

NCAR Interactive should operate on the latest versions of operating systems that are available upon release. Currently these include:

- SunOS (4.0.3 and 4.1),
- ULTRIX (4.2)
- UNICOS (6.1)
- AIX (3.1)
- IRIX (3.3)

NI should use the latest versions of X11 and Motif which are currently R5 and 1.1.3 respectively.

The following input data format libraries should be used: netCDF, HDF, CDF, ascii, FORTRAN, and C binary. Perhaps FITS, VIFF (KHOROS), and AVS fields should be supported.

For output, the main format will be CGM. Translation to HDF, Sun raster, NRIF, and X Window dump formats will be supported.

SECTION 3

HARDWARE REQUIREMENTS

NCAR Interactive should port to the following systems which are currently supported for NCAR graphics:

- CRAY Y-MP and CRAY X-MP
- Digital Equipment DECstation
- IBM RISC System/6000
- Silicon Graphics IRIS 4D-series
- Sun-3, Sun-4, and Sun SPARCstation
- HP 9000 series 700 workstations
- Any systems to be added to this list for the Version 4.0 release of NCAR Graphics should also be supported.

Hardware features of high-end workstations, which many of our users may not have, should be a low priority. Since the primary output mechanism will still be CGM and NRIF files, there are no output hardware requirements from the point of view of the NCAR Interactive package.

SECTION 4

New programming interface to NCAR Graphics (High Level Utilities)

4.1 Overview

The High Level Utilities (HLUs) being created as a new interface to NCAR Graphics will not replace the current Fortran interface. The existing Fortran interface wil continue to be supported. The new interface will facilitate the development of NCAR Interactive. However, this newinterface also has a number of advantages to the end user.

One of the dominant user comments is "NCAR Graphics is hard to learn and use." The reason NCAR Graphics is hard to learn and use is that the various utilities do not have a consistent interface, memorization is often required to know which functions to call in what order, there is too much room for undetected user error, and NCAR Graphics does not accommodate users of multiple skill levels. In fact, current users must be NCAR Graphics experts to produce the simplest data plots.

The primary goal in designing the HLUs is to provide a consistent visualization model to the user. Standard sets of configuration parameters, called resources, will be developed for each HLU and, where applicable, different utilities will share resource names to provide consistency. Objects like titles, tick marks, labels, legends, and label bars will be named consistently.

These sets of configuration parameters will provide much of the capability of the current version of NCAR Graphics utilities, but some of the configurability of an existing utility may not be included in some cases. Users who have extremely specialized needs will still have to use the existing Fortran interface, but users who want to create visualizations easily will be able to do so. The HLUs will include the most commonly used features of the current utilities. The HLUs will not require any changes to existing user code. @@@verify!!!

The HLUs will also support the demands of interactive programs. Functions for retrieving graphics state information from an HLU will be provided. For example, every 1D and 2D HLU will be able to transform normalized device coordinate (NDC) pairs into data space coordinate pairs. Also, every HLU will be able to determine if an NDC pair is in its viewport area. These types of functions will facilitate "point-and-click" application development.

4.2 HLU usage

The number one change in the programming interfaces to NCAR Graphics is to introduce the concepts of abstraction and information hiding. The HLUs employ the abstraction that every plot is an object that is created and can be manipulated through a series of argument-setting routines. This allows the HLUs to hide time-consuming data control tasks from the user. For those familiar with X Window programming, the mechanisms for creating and specifying plots will be similar, but far simpler.

The current version of NCAR Graphics treats the data plot as the result of a sequence of function calls. This is not an intuitive model, and users do not learn it easily. It often leaves the user guessing at the way a function call affects the output, and it makes programs extremely hard to alter or extend. Furthermore, current NCAR Graphics requires the user to

manage various internal data structures like work arrays and common blocks. For example, a difficult abstraction is the area map needed by the Areas utility. The user must specify how big this array is, initialize it, and coordinate different utilities that draw into it. This is a very complex task, and it represents a lot of overhead for beginners.

The new HLUs will manage the graphics state of NCAR Graphics on behalf of the user. With the new HLUs, there will be only a few standard functions needed by the user to manipulate the graphics state. The names of these have not been defined, but the following is a list of these functions:

init	initializes the graphics device.
create	creates an instantiation of a plot style.
loaddef	loads defaults from a user-specified defaults file. This allows for plot configuration outside of the program.
setarg	used to set all single-value resources for a plot style.
arraysetarg	used to set resources that require arrays.
getarg	used to retrieve values of resources for a plot style.
arraygetarg	used to retrieve values of resources that are arrays.
draw	draws a plot onto the output frame.
destroy	removes an instantiation of a plot style from the graphics state.
overlay	assigns one instantiated plot to be the overlay of another.
update	erases and redraws a graph without advancing the frame.
frame	erases current output frame and prepares a new one.
end	closes the graphics device.
load_defaults	loads a user-specified default resource file.
dump_defaults	writes all current resources for a given plot style into a defaults file.

These functions will be used as follows: *init* must be called at the beginning of the user's program. The user can then load a defaults file with the *loaddefs* function (the syntax of the defaults file is covered later). The defaults file has two main

uses. First, it lets the user customize the default parameters used when plots are created, and second, the user can configure the output of a program by editing the defaults file without recompiling the program.

The user can then *create* instantiations of the plot styles he wants to use. *Create* returns a unique id that represents the instantiation of the requested plot style. Each plot id must be thought of as a unique data plot that "knows" how to draw itself based on its input resources.

Next, the user can alter the default configuration of the plot style by calling the *setarg* function with three pieces of information as parameters: the plot id, the resource name, and the resource value. The *setarg* function only alters the state of the plot whose plot id was passed to it. The *arraysetarg* sets resources that are arrays. This is fundamentally different than traditional NCAR Graphics because arguments don't affect a global graphics state but only the graphics state of the instantiated plot.

The *overlay* function assigns one instantiated plot to a base plot. When a plot is tagged as an overlay, it inherits the data and viewport transformations from the base plot. In some circumstances, some of the overlay plot's resources may be pre-empted by the resources of the base plot.

The *display* function draws a plot onto the output frame. The *display* function takes a plot id and draws the plot described by the resources assigned to that plot id. If the plot selected is a base for any other plots, the other plots are drawn at the same time on top of the base plot.

The *destroy* function removes a plot id from the current list of plot ids and frees any storage space used by the plot. *Update* forces a redraw of a plot. However, the entire plot is not recomputed; only the portion that has been altered or changed.

If nothing has changed, the plot is merely redrawn. *Frame* erases the current output frame and prepares a new one. *End* closes the graphics device.

The *load_defaults* function allows the user to load a predefined set of default resources to use when creating plots. The *dump_defaults* function creates a defaults file for any given plot id. The defaults file can then be used again to create the same configuration for the given plot style.

The following is a pseudocode example of what a user program might look like to create a contour plot with a map using default color and contour specifications. Please note that the data-reading functions are commented as user defined functions. Programmers using the HLUs are still responsible for reading, writing, and processing data themselves. If you are interested in the built-in data handling capability, please refer to the "NCAR Command Language Specification."

```
init(X11);                          /* specifies X11 output */
load_defaults("~ethan/defaults");   /* loads default resources */
pid = create(CONTOUR);              /* instantiates contour plot */
setarg(pid,Nmp,True);               /* turns map on */
setarg(pid,NmpOutlineType,PS);      /* sets outline style */
setarg(pid,NmpProjection,OR)        /* sets projection style */
zdat = readmydataone();             /* user routine that reads data */
setarg(pid,NcnXdatamin,0.0);        /* sets data coordinate boundary */
setarg(pid,NcnXdatamax,180.0);      /* sets data coordinate boundary */
setarg(pid,NcnYdatamin,0.0);        /* sets data coordinate boundary */
setarg(pid,NcnYdatamax,90.0);       /* sets data coordinate boundary */
setarg(pid,NcnZdat,zdat);           /* sets data array */
setarg(pid,NcnNumX,30);             /* sets dimensionality of data */
setarg(pid,NcnNumY,40);             /* sets dimensionality of data */
setarg(pid,NtiMain,"DATASET 1");    /* sets title for frame one */
draw(pid);                          /* draws plot on screen */
frame();                            /* advances the frame */
zdat = readmydatatwo();             /* user defined data-reading func.*/
setarg(pid,NcnZdat,zdat);           /* resets the data array */
setarg(pid,NtiMain,"DATASET 2");    /* sets title for frame two*/
draw(pid);                          /* draws new plot */
frame();                            /* advances final frame */
end();                              /* closes graphics device */
```

This example creates two frames of output. After the plot id "pid" has been configured, the next frame only needs to change the resources that have changed—the title and the data array in this case. The two data arrays are assumed to be the same dimension.

The preceding example also demonstrates how the HLUs work. A plot id is returned after the plot is created. Now the configuration of that plot can be changed at will by calling **setarg** and providing the plot id, the resource name, and the resource value to the **setarg** function. Since the map projection did not change between **draw** calls in the example, the second **draw** does not recompute the map projection; it merely uses the map segment that was computed during the first **draw** call. The second draw call only computes the new contours from the new data and sets the new title string in the appropriate place.

Another item demonstrated in the example is the **load_defaults** function. This function reads in a user defaults file. A user defaults file can contain custom default resources. Essentially every resource that can be set with a **setarg** function can be set in a defaults file. Users can develop sets of defaults files for different visualization applications. Furthermore, resources that are set in a defaults file can be changed without recompiling the application. This allows users to change everything from fonts to colors without recompiling their code. The user defaults file is described later in this section.

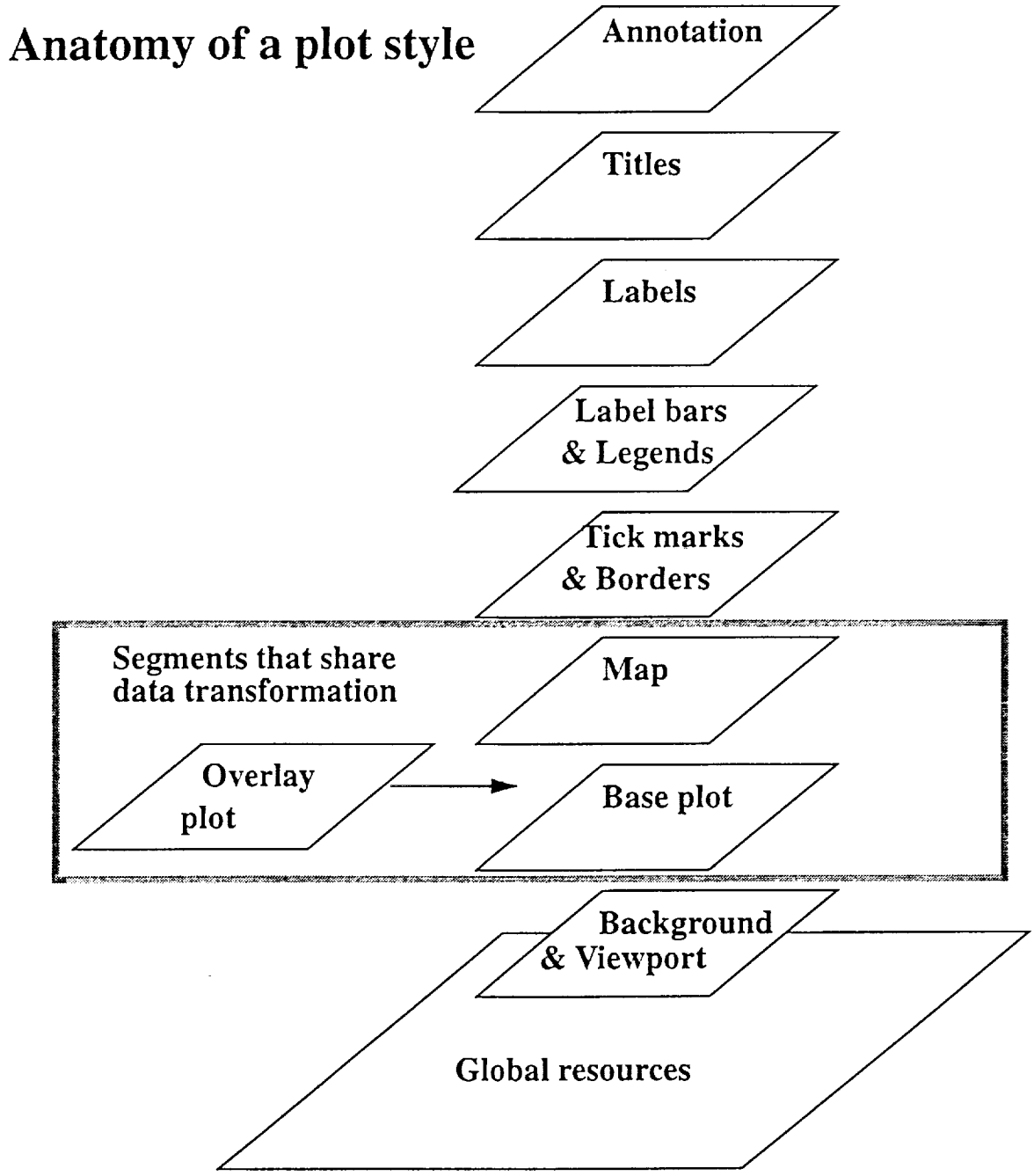
This HLU section gives a very brief description of how the HLUs are different from current NCAR Graphics and indicates how they will function. A more in-depth discussion of the utilities and their resources follows.

The concept of a resource is fundamental to understanding the HLUs. In the previous paragraphs, the resource was described as a configuration parameter. Essentially, a resource is a piece

of information used by an HLU to determine how to configure its output. As the example showed, resources can control a variety of configurations. Some resources like fonts, lines, colors, fill patterns, tick marks, titles, tick mark labels, and map settings are set the same and behave the same across similar HLUs. Others, such as resources that specify contour levels for contour plots and resources that set line styles for X-Y plots are different and specific to each HLU.

This HLU section is organized into four parts. The first describes a global resource set. The second describes common basic resources, which are basic objects like fonts, lines, and symbols. The third part discusses common composite resources. These are resources that are built up of the basic objects like tick marks, labels, maps, and titles. The final part describes each of the HLUs in detail, as well as each of the HLU-specific resources. The next figure provides a model to help you visualize the organization of the composite resources and the HLU.

FIGURE 2 Anatomy of a Plot Style



4.3 HLU efficiency

This information is incidental to the current section. It is included because initial reviewers of this functional spec expressed many concerns about efficiency.

TThe efficiency of the HLUs is a very important concern if the HLUs are to adaquetely support the demands of interactive graphics. Initially, the HLUs will be implemented using the NCAR Graphics 3.0 utilities. Because of this, much of the efficiency is bound by the efficiency of the low-level implementation of NCAR Graphics. However, initial tests show many areas where the HLUs could enhance the efficiency of NCAR Graphics.

The most significant advance in efficiency is gained by segmenting the plot produced by an HLU into the groups listed in Figure 2 on the preceding page. This enables one group to be changed and redrawn without having to recompute the remaining groups. For example, tests show that sub-second response time can be achieved for updating every group except the maps and the base plot of an HLU. This means that changing tick mark style, fonts, labelbars, and legends can be done very quickly.

Other tests show that most of the execution time of an NCAR Graphics program is in the Areas utility. The most common way to slow down an NCAR Graphics program is to only use one area map and draw everything into it. The algorythms used by Areas slow down exponentially as the number of elements in area map grow. Separating the plot into several small area maps speeds execution. In a test example, a color-filled contour plot with a map masking out the land was drawn in two-fifths of the time simply by separating the contour areas from the map areas.

4.4 Resource naming conventions

Resources are grouped by function, so their naming convention reflects this grouping. There are three kinds of resources: common resources, common composite resources, and plot-specific or HLU-specific resources. Common resources are common objects used throughout the HLUs; they cannot be set by users. For example, a common resource is **Font**—it does nothing by itself. However, when joined with **Nti** (the titling composite resource) and **Main** (the main title of the HLU) to form **NtiMainFont**, this new resource sets the font for the main title of its HLU. The following list shows how the **Font** common resource is used in different common composite resources.

NtmLabelFont	Sets font for tick mark labels.
NlgTitleFont	Sets font for legend title.
NlbTitleFont	Sets font for label bar title.
NlgLabelFont	Sets font for labeling items in a legend.

The names of common resources are outlined in section 4.7 and composite resources are outlined in section 4.8. Table 1 below lists the three character codes used to identify the group to which a resource belongs. The three character codes for identifying HLU-specific resources are listed in Table 2.

TABLE 1 Resource naming conventions for common composite resources

Prefix	Resource class
Nan	Annotation
Nti	Title
Ntm	Tick Mark
Nlb	Label Bar
Nmp	Map
Nbk	Background
Nvp	Viewport
Nlg	Legend
Nbo	Border
Ngb	Global

TABLE 2 Resource naming conventions for High Level Utilities

Prefix	HLU name
Ncn	Contour
Nvr	Vector
Nsr	Surface
Nis	Iso-Surface
Nsl	Streamline
Npp	PolyPaint
Nxy	X-Yplot
Nhs	Histogram
N3d	Common Surface and Iso-surface

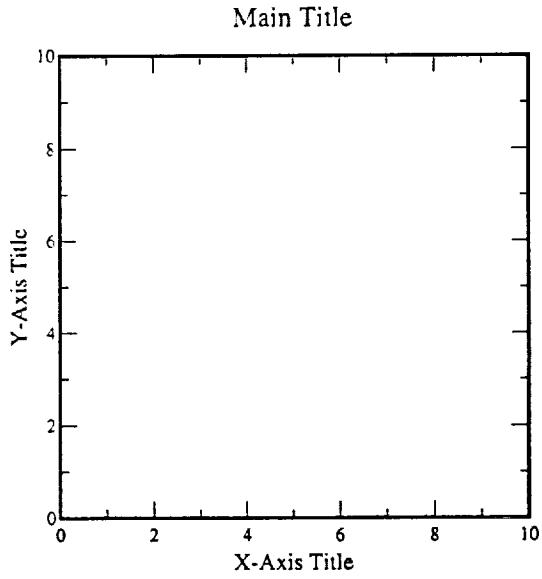
4.5 User defaults file

A function will be available to allow users to load a defaults file when executing their program. The defaults file can contain any of the resources for any of the plot styles. The defaults are used whenever a plot is instantiated with the create function. The user can override the defaults by changing the specific resource with the **setarg** function, but the user must then recompile the program. By changing resource values in the user defaults file, small changes in plot configuration can be made without recompiling the program. Users can set the resources in the user defaults file by entering the resource name followed by a colon and the value for the resource. The following is what a default resource file might look like for an X-Y plot.

```
Nvpx                    : .10
Nvpy                    : .90
NvpWidth                : .80
NvpHeight               : .80
NtiMain                 : Main Title
NtiMainFont             : Times-Roman
NtiMainFontSize         : .04
NtiX                    : X-Axis Title
NtiXFont                : Times-Roman
NtiY                    : Y-Axis Title
NtiYFont                : Times-Roman
NtiXFontSize            : .03
NtiYFontSize            : .03
NtmXGroup               : True
NtmYGroup               : True
NtmXMajorLength         : .009
NtmXMinorLength         : .0045
NtmXMinorPerMajor       : 1
NtmXLabelFont           : Times
NtmXLabelFontSize       : .02
NtmYMajorLength         : .009
NtmYMinorLength         : .0045
NtmYMinorPerMajor       : 1
NtmYLabelFont           : Times-Roman
NtmYLabelFontSize       : .02
NtmXStyle               : Linear
```

```
NtmYStyle          : Linear
NtmXMode           : Manual
NtmYMode           : Manual
NtmXStart          : 0
NtmYStart          : 0
NtmXEnd            :10
NtmYEnd            : 10
NtmXSpacing        : 2
NtmYSpacing        : 2
```

When it is loaded, this example default resource file would configure the viewport, titles, and tick marks for any 1-D or 2-D plot. The following figure is what a plot, created using the above resources, might look like.



4.6 Global resources

Global resources are resources used by all instantiated plots. A special plot id value will be assigned to these resources. Users set global resources with the standard NCAR Interactive resource-setting procedure.

4.6.1 Color

A color map of 256 colors is allowed by NCAR Graphics. The HLUs will have the same restriction. Each of these colors will be indexed from 0 to 255. The currently planned color model is RGB. However, HLS and some other color models should be made available as soon as possible. Whenever a color index is set in any plot resource, the index refers to the global color map.

A set of default color maps will be made available when the HLU library is released. These default color maps are in the form of a user defaults file; it can be loaded just like any other defaults file.

Support for 24-bit color should be added as soon as possible.

The following color resources will be included:

4.6.1.1 NgbColor

Switches between indexed color and gray scale colors.

4.6.1.2 NgbColorMap

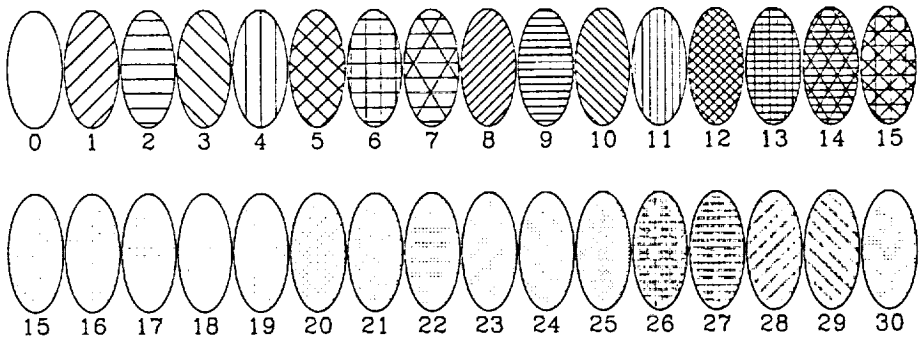
A color map can be set by passing an array of red, green, and blue as the argument to this resource.

4.6.1.3 NgbColorMapLen

Specifies the number of colors in the color map that is loaded.

4.6.2 Fill patterns

Users will be able to choose from a set of predefined fill patterns. Fill patterns can be chosen with an index number, the same way colors are chosen. Users can select an indexed fill pattern or create their own and add them to the fill pattern list. Areas, like contours, can be assigned both a color and a fill pattern.



"FILL INDEX"

The predefined set of fill patterns are indexed, and whenever a fill pattern is set, an index number is placed into the global fill pattern table.

The following fill pattern resources will be included:

4.6.2.1 NgbFillMap

The global resource that contains all valid fill patterns.

4.6.2.2 NgbFillMapLen

The number of currently defined fill patterns in the fill map.

4.7 Common basic resources

This section describes objects that are common to all utilities. These objects include text, color, lines, and fill patterns. These objects and their resources are used by all of the HLUs and by composite resources. Names are given to these resources, but they may change when a resource-naming convention is adopted. Furthermore, these names reflect how the C binding interface will look. The Fortran bindings will be developed after a complete specification of the resources is finished

When a basic object is used by a composite object, the name of the composite object is inserted into the resource name. For example, LineThickness becomes GridLineThickness when setting the thickness for the composite object grid.

4.7.1 Text

Text is a basic resource that uses subresources to configure the output. The following list of text resources includes a description of the information needed to set them.

4.7.1.1 Font

The output font can be selected by passing a two-character code representing the font type. Currently the fonts can be:

HERSHEY:CARTOGRAPHIC_ROMAN
HERSHEY:CARTOGRAPHIC_GREEK
HERSHEY:SIMPLEX_ROMAN
HERSHEY:SIMPLEX_GREEK
HERSHEY:SIMPLEX_SCRIPT
HERSHEY:COMPLEX_ROMAN
HERSHEY:COMPLEX_GREEK
HERSHEY:COMPLEX_SCRIPT
HERSHEY:COMPLEX_ITALIC
HERSHEY:COMPLEX_CYRILLIC
HERSHEY:DUPLEX_ROMAN
HERSHEY:TRIPLEX_ROMAN
HERSHEY:TRIPLEX_ITALIC
HERSHEY:GOTHIC_GERMAN
HERSHEY:GOTHIC_ENGLISH

HERSHEY:GOTHIC_ITALIAN
HERSHEY:MATH_SYMBOLS
HERSHEY:SYMBOL_SET1
HERSHEY:SYMBOL_SET2

4.7.1.2 FontSize

The size of the font is a very tricky thing to specify. For simplicity, only one font sizing model is used in the HLUs. The size for a font is expressed as a fraction of the size of the output frame coordinates. This is how fonts will be sized unless the text being drawn is part of a composite resource.

4.7.1.3 FontAspect

A font's aspect ratio specifies a character's width with respect to its height. A value of one means that the height and width of each character are equal. A value of .5 means that the width of each character is half its height.

4.7.1.4 FontThickness

Since all NCAR Graphics fonts are stroked fonts, a line thickness must be set. Just as the font size is represented as a fraction of the output frame coordinates, the line thickness is also set this way.

4.7.1.5 FontColor

Each HLU shares the color map defined as a global resource.

4.7.1.6 TextJust

Text justification does not have anything to do with fonts, but it is an attribute of setting text. Below is a picture with a rectangle drawn around a piece of text. This rectangle is called

the text bounding box. There are nine points on the box: one at each corner, one at the midpoint of each edge, and one in the center. These are the points about which text can be justified.

4.7.1.7 TextPosX & TextPosY

When text location is not automatically set by the utility, a coordinate pair must be specified to set the text. The justification point set with TextJust is positioned at these coordinates. These coordinates are in output frame coordinates.

4.7.1.8 TextAngle

This is an angle, expressed in degrees, by which the text is rotated around the justification point.

4.7.1.9 TextItem

This is the string that is printed onto the output frame. This text string uses the same function codes that have been used in NCAR Graphics in the past. Therefore, a great deal of configurability of the output string is possible. Function codes are discussed in the Plotchar section of the NCAR Graphics, Version 3.00 manual.

4.7.2 Lines

Lines are a bit simpler than fonts. Lines can have color, thickness, dash pattern, and smoothness. The following line resources will be included:

4.7.2.1 LineThickness

Set as a fraction of the size of the output frame.

4.7.2.2 LineDashPattern

Controls the sequence of dash characters used to draw the line. The dash pattern is set in the same way as the Dashchar utility described in the NCAR Graphics Version 2.00 manual.

4.7.2.3 LineDashLength

Controls the length of each dash. This resource is set as a fraction of the output frame size.

4.7.2.4 LineColor

Sets the color of the line in the same way as FontColor.

4.7.2.5 LineSmooth

Turns on and off the line smoothing features described in the NCAR Graphics Version 2.00 manual.

4.7.3 Arrays

Several resources are arrays used throughout the HLUs. There are two ways users can set these resources. One is to use the **arraysetarg** function discussed previously. The other way is to use a combination of two resources. One resource is the resource name of the array and the other is an index into that array. Every array resource has a specific resource with the word "Index" appended to it. This resource can be set to an index into the main resource being set. For example, the X-Y plot allows the user to have multiple data curves. Each of these curves can have a unique dash pattern. To set the first curve's dash pattern, the NxyLineIndex resource is first set to 1; when the NxyLineDashPattern resource is set, it is set for the first curve only.

4.7.4 Background color

One resource will be included to control background color.

4.7.4.1 NbkColor

Specifies the background color for the plot.

4.7.5 Size and viewport

The following resources will be included to control size and viewport.

4.7.5.1 NvpX and NvpY

Sets the X and Y coordinates for the upper left corner of the viewport in output frame coordinates.

4.7.5.2 NvpWidth and NvpHeight

Sets the width and height of the viewport in output frame coordinates.

4.8 Common composite resources

Common composite resources are resources located in more than one HLU; they are made up of basic resources, lines, text, and colors. Tick marks, tick mark labels, grids, legends, label bars, titles, and maps are all examples of common composite resources.

4.8.1 Tick marks, tick mark labels, and grids

The most common types of tick marks are tick marks for 1D and 2D plots. All of the tick mark resources listed here are for both the X and the Y axis unless otherwise stated.

Each axis can also be divided into sides, (i.e. top, bottom, right, and left). The resources listed in this section are generic, so the top, bottom, left, right and axis have been removed from the resource name. In reality, each resource is prepended with this information. For example, the TickMarkSpacing resource is really four resources called XTTickMarkSpacing, XBTickMarkSpacing, YRTickMarkSpacing, and YLTickMarkSpacing. Separating the resources in this way allows users a great deal of flexibility as they configure the tick marks of their plots.

Since composite resources are generated by an HLU, all of the sizes for text and lines are relative to the maximum viewport range. This is done so that sizes can be more easily judged by the user. Furthermore, the tick mark border is drawn around the viewport and the tick marks share the data transformation of the base plot, or HLU. When tick mark resources accept data values, the underlying data transformation of the HLU is used to map the value to screen coordinates.

There are many ways tick marks can be configured. This section covers resources that affect the output of all styles of tick marks first; it then covers resources specific to certain styles of tick marks. These styles are log, linear, time, and geographic coordinates.

Note that all of the resources in this section apply only to plots produced **without** maps. Grids, tick marks, and labels for plots using one of the mapping transformations are covered in the Maps section of this document.

To disable tick marks completely, set the TickMode resource to NONE.

NtmXGroup & NtmYGroup	Groups top and bottom, and right and left axes together.
NtmMajorLineThickness **NtmMajorLineColor**	Sets line attributes for major tick marks.

NtmMinorLineThickness **NtmMinorLineColor**	Sets line attributes for minor tick marks.
NtmMajorLength & **NtmMinorLength**	Sets the major and minor tick mark length.
NtmMinorPerMajor	Sets the number of minor tick marks per major tick mark.
NtmLabelFont **NtmLabelFontSize** **NtmLabelFontAspect** **NtmLabelFontColor** **NtmLabelTextJust** **NtmLabelFontThickness** **NtmLabelTextAngle**	Set the text attributes for the tick mark labels.
NtmLabelOffset	Displaces the tick mark label justification points from their default position.
NtmBorder **NtmBorderLineDashPattern** **NtmBorderLineColor** **NtmBorderLineDashLength** **NtmBorderLineThickness**	Defines border style.
NtmAxisControl	Defines how tick mark axes are drawn.
NtmAxisLocation	Displaces the tick mark axis up or down, or left or right.
NtmMajorGrid **NtmMajorGridLineDashPattern** **NtmMajorGridLineDashLength** **NtmMajorGridLineColor** **NtmMajorGridThickness**	Draws a grid connecting major tick marks.
NtmMinorGrid **NtmMinorGridLineDashPattern** **NtmMinorGridLineDashLength** **NtmMinorGridLineColor** **NtmMinorGridLineThickness**	Draws a grid connecting minor tick marks.
NtmMajorOutwardLength	Specifies portion of major tick mark to be drawn outward.
NtmMinorOutwardLength	Specifies the portion of the minor tick marks to be drawn outward.
NtmLabelFormat	Set the numeric format for the tick mark labels.
NtmLabelFormatExponent	Sets exponent when scientific notation is used.
NtmLabelFormatFraction	Sets fraction when scientific notation is used.
NtmStyle	Selects between log, linear, geographic, and time tick mark styles.
NtmMode	Selects between none, automatic, manual, and explicit tick mark modes.
NtmMaxTicMarks - Automatic Mode	Sets the maximum number of allowable tick marks.

NtmStart - Manual Mode	Sets the starting data value for tick marks.
NtmEnd - Manual Mode	Sets the ending data value for tick marks.
NtmSpacing - Manual Mode	Sets the interval between major tick marks.
NtmSpacingType - Manual Mode	Selects between three different ways of using TickSpacing.
NtmMinorStart - Manual Mode	Sets the starting data value for minor tick marks.
NtmMinorEnd - Manual Mode	Sets the ending data value for minor tick marks.
NtmValues - Explicit Mode NtmValueIndex	Sets data value for each tick mark.
NtmNumTicks - Explicit Mode	Sets the number of tick marks total.
NtmLabels - Explicit Mode NtmLabelIndex	For each tick mark specified in TickMarkValues, this sets a tick mark label string to use.
NtmMappingOrder	Specifies if axis is increasing or decreasing.
NtmGeoTickStyle	Selects between various ways of labeling latitude by longitude coordinates.
NtmGeoTickAxis	Sets the axis to either latitude or longitude.
NtmTimeTickStyle	Selects between various ways of labeling time tick marks.
NtmTimeDataBase	Sets the starting time of the axis.
NtmTimeDataSpacing	Sets the time interval between data points.
NtmTimeTickStart - Manual	Sets the starting time of the first tick mark.
NtmTimeTickEnd - Manual Mode	Sets the ending time of the last tick mark.
NtmTimeTickSpacing - Manual Mode	Sets the time interval between tick marks.
Explicit Mode Time Ticks	Should be analogous to Explicit mode for regular tick marks.

4.8.2 Legends

This section describes the resources that specify a legend. Legends, like label bars (described below), are used as a key to accompany a plot. Legends are very similar to label bars, but they differ in that they define line representations (dash patterns) and symbols rather than fill patterns. Legends can be displayed vertically or horizontally and labels may appear on any side.

The base plot to which the legend belongs provides all of the symbol and line information automatically. For example, an X-Y plot with five curves and five dash patterns will automatically draw those five dash patterns and label them with "curve A," "curve B," etc., if the NlgLegend resource for the X-Y plot is turned on. Most of the legend resources are set automatically by the HLU.

If the level of configurability is not adequate, the Annotation HLU can create custom legends with complete configurability by overriding the automatically set resources. Otherwise, the dash patterns and symbols will be drawn in the order they are used by the HLU. Tables 3 and 4 show resources that must be set by the user and that are set automatically by the base plot.

TABLE 3. Resources not set by the base plot

NlgLegend	Turns legends on and off.
NlgOrientation	Sets vertical or horizontal ordering of legend elements.
NlgSymbolOrientation	Sets vertical or horizontal symbol or line.
Nlgx **Nlgy** **NlgWidth** **NlgHeight**	These four resources set the coordinates of the upper left corner, and the width and height of the legend bounding box in output frame coordinates.
NlgLabelText **NlgLabelFont** **NlgLabelFontSize** **NlgLabelFontThickness** **NlgLabelFontAspect** **NlgLabelTextAngle** **NlgLabelFontColor**	Sets the string and text attributes for each legend element.
NlgLabelPosition	Places legend labels above, below, to left, or right of legend symbol or dash.

TABLE 3. Resources not set by the base plot

NlgTitleText NlgTitleFont NlgTitleFontSize NlgTitleFontThickness NlgTitleFontAspect NlgTitleTextAngle NlgTitleFontColor	Sets string for legend box title and associated text attributes.
NlgDrawBorder NlgBorderLineDashPattern NlgBorderLineDashLength NlgBorderLineColor NlgBorderLineThickness	Draws the bounding box arround legend.

TABLE 4. Resources set automatically by the base plot

NlgElementType NlgNElements	Each item displayed in the legend can be either a symbol or a line. This resource keeps track of which is which and the total number of elements.
NlgIndex	Selects current legend element when using the index scheme for setting resources.
NlgLineColor NlgDashLineLength NlgLineThickness NlgDashPattern	These resources are used when an element is a line.
NlgSymbol NlgSymbolColor NlgSymbolScaleFactor NlgSymbolFillPattern	These resources are used when an element is a line symbol.

4.8.3 Label bars

This section describes the resources that specify a label bar object. Label bars are used as a key or legend to accompany a plot. They consist of a rectangular, labeled bar that can be colored and filled with a solid color or patterned lines. The bar may be displayed vertically or horizontally, and labels may appear on any side.

As with the legend resources, the color, number of boxes, and the labels for each box are set by the HLU. Custom label bars can be generated through the Annotation HLU. Table 5 lists the label bar resources.

TABLE 5. Resources that specify label bars

NlbLabelBar	Turn label bar on and off
NlbOrientation	Horizontal or vertical label bar.
NlbX NlbY Nlbwidth Nlbheight	Sets bounding box for label bar.
NlbAreaWidth NlbAreaHeight	Sets percentages of Nlbwidth and NlbHeight where labelbar will be drawn.
NlbLabelText NlbLabelFont NlbLabelFontColor NlbLabelFontSize NlbLabelFontAspect NlbLabelTextAngle	Sets strings for label bar labels and associated text attributes.
NlbAlignment	Places labels in middle of box, at left corner, or at right corner.
NlbLabelPosition	Places labels at top, bottom, right, or left of the label bar.
NlbBoxLineColor NlbBoxLineThickness NlbBoxLineDashPattern NlbBoxLineDashLength	Sets the line attributes for boxes surrounding label bar boxes..
NlbTitleText NlbTitleFont NlbTitleFontSize NlbTitleFontColor NlbTitleFontThicknes NlbTitleFontAspect	Sets string and associated text attriubtes for label box title.
NlbDrawPerim NlbPerimLineThickness NlbPerimLineDashPattern NlbPerimLineDashLength NlbPerimLineColor	Draws a border around label bar bounding box.

TABLE 6. @@@

NlbNboxes	Number of current label bar boxes.
NlbBoxFillPattern **NlbBoxIndex** **NlbBoxColor**	Fill pattern and color for each box.

4.8.4 Titles

Titles are special kinds of text items in which the position of the text is defaulted to be at some fixed location above, below, right, or left of the plot. The locations of X and Y titles will be guaranteed not to interfere with the tick mark labels. Resources for small adjustments up or down and left or right are provided, as well as a resource for flipping the rotation of a title. This is particularly useful for the Y axis title. It is often a matter of taste how the Y axis title is positioned.

TABLE 7. @@@

NtiMainText **NtiMainFont** **NtiMainFontSize** **NtiMainColor** **NtiMainTextJust** **NtiMainFontAspect** **NtiMainFontThickness** **NtiMainTextAngle**	Sets the main title of a plot.
NtiMainXOffset **NtiMainYOffset**	Displaces title from its default position.
NtiMainPosition	Title can be placed on left, center, or right portion of plot.
NtiYAxisText **NtiYAxisFont** **NtiYAxisFontSize** **NtiYAxisColor** **NtiYAxisTextJust** **NtiYAxisFontAspect** **NtiYAxisFontThickness**	Controls Y-axis title.
NtiYAxisXOffset **NtiYAxisYOffset**	Displaces Y-axis title from its default location.

TABLE 7. @@@

NtiYAxisPosition	Y-axis title can be on left or right side of plot.
NtiXAxisText NtiXAxisFont NtiXAxisFontSize NtiXAxisColor NtiXAxisTextJust NtiXAxisFontAspect NtiXAxisFontThickness	Controls X-axis title.
NtiXAxisXOffset NtiXAxisYOffset	Displaces X-axis title from default location.
NtiXAxisPosition	X-axis title can be on top or bottom of plot.

4.8.5 Maps

Maps can only be used in the 2D utilities Contour, Vector, and Streamline. In these utilities, maps can either be on or off.

Nmp	Turns on the mapping option.
NmpProjection	Selects the mapping projection.
NmpTicks NmpTickLength NmpTickThickness NmpTickColor	Controls map tick mark attributes.
NmpTickLabels NmpTickLabelFont NmpTickLabelFontColor NmpTickLabelFontSize NmpTickLabelFontAspect NmpTickLabelFontThickness NmpTickLabelTextAngle NmpTickLabelTextJust	Turns on map tick mark labels.
NmpTickLabelStyle	Chooses between degrees, degrees-minutes, and degrees-minutes-seconds display styles.
NmpGrid	Sets whether or not to turn on grid (lat and lon lines).
NmpGridDotSpace	Sets the distance between points used to draw a grid.
NmpGridSpace	Sets the grid spacing in degrees.
NmpUserLinesDotDist	Sets the distance between dots along a dotted line drawn by MAPIT.
NmpPlotterResltn	Sets the width of the target plotter in plotter units.
NmpUserLinesDashPattern	Sets the dash pattern for lines drawn by the MAPIT utility.

NmpUserLinesType	Sets the type of line (dotted or dashed) drawn by MAPIT.
NmpPerimLineDashPattern NmpPerimLineDashLength NmpPerimLineColor NmpPerimLineThickness	Controls how perimeter line is drawn.
NmpGridLineDashPattern NmpGridLineDashLength NmpGridLineColor NmpGridLineThickness	Controls how map grid line is drawn.
NmpLimbLineDashPattern NmpLimbsLineDashLength NmpLimbsLineColor NmpLimbsLineThickness	Controls how limb line is drawn.
NmpCOLineDashPattern NmpCOLineDashLength NmpCOLineThickness NmpCOLineColor	Controls how continental outlines (CO) are drawn.
NmpUSLineDashPattern NmpUSLineDashLength NmpUSLineThickness NmpUSLineColor	Controls how US outlines are drawn.
NmpPOLineDashPattern NmpPOLineDashLength NmpPOLineDashThickness NmpPOLineColor	Controls how political outlines are drawn.
NmpLimbLine	Sets whether or not lim lines should be drawn.
NmpOutlineType	Sets the type of outlines to use (none, continental, state, etc.).
NmpOutlines	Sets whether or not outlines are dotted or dashed.
NmpElliptical	Sets whether or not a map is inscribed within an ellipse.
NmpLabels NmpLabelsFont NmpLabelsFontSize NmpLabelsFontColor NmpLabelsFontThickness NmpLabelsFontAspect	Sets whether or not to label the meridians, the poles, and the equator
NmpMinVectorLength	Points closer than this value to a previous point are omitted
NmpPerim	Sets whether or not to draw a perimeter
NmpSatDistance	Sets how many Earth radii the satellite is from the Earth's center
NmpSatSight2Center	Measures the angle between the line to the Earth's center and the line of sight
NmpSatU2Projection	Measures the angle from the positive U axis to the line OP, where O is the origin, and P is the projection of the desired line of sight

NmpProjectionOrigin	The coordinates of the origin of projection (latitude, longitude)
NmpProjectionRotation	Angle of rotation of a projection
NmpRectLimitType	Sets howto interpret the rectangular limits of a map
NmpRectLimit1 **NmpRectLimit2** **NmpRectLimit3** **NmpRectLimit4**	The rectangular limits of a map. These values depend on the NmpRectLimitType.
NmpBoundMaskType	Sets whether or not to mask areas in NmpBoundMask when drawing boundaries
NmpBoundMask	Array of area identifiers to mask or unmask
NmpBoundMaskSize	Size of the NmpBoundMask array
NmpGridMaskType **NmpGridMask** **NmpGridMaskSize**	Sets whether or not to mask areas in NmpGridMask when drawing grid lines
NmpLineMaskType **NmpLineMask** **NmpLineMaskSize**	Sets whether or not to mask areas in NmpLineMask when drawing lines with the MAPIT utility
NmpAreaFillType	Sets whether or not to mask areas in NmpAreaaFill when filling areas
NmpAreaFill	Array of area identifiers to mask or unmask
NmpAreaFillSize	Size of the NmpAreaFill array
NmpAreaFillColor	Array of color table indices specifying the area fill color
NmpAreaFillPattern	Array of fill flags which determines an area's fill pattern
NmpLatLabels **NmpLonLabels**	Sets whether latitude and longitude labels will be along a constang grid line or at the perimeter of a plot
NmpLatLabelConstant **NmpLonLabelConstan**	Set the latitude and longitude for labeling a grid
NmpLatLabelStyle **NmpLonLabelStyle**	Sets the format for latitude and longitude labels
NmpLatLabelSpace **NmpLonLabelSpace**	Sets the spacing in degrees between labels
NmpLatLabelFont **NmpLatLabelFontSize** **NmpLatLabelFontThickness** **NmpLatLabelColor** **NmpLatLabelFontAspect** **NmpLonLabelFont** **NmpLonLabelFontSize** **NmpLonLabelFontThickness** **NmpLonLabelColor** **NmpLonLabelFontAspect**	Sets the text characteristics of the grid labels for latitude and longitude

4.9 X-Y plot

An X-Y plot is a two-dimensional plot of zero or more curves. The functionality is similar to that of the NCAR Graphics Autograph utility. An X-Y plot can have any number of curves. For each curve, a line style, color, pattern, etc. must be specified or defaulted. These attributes are passed as arrays of attributes. The size of the array arrays is determined by the number of curves in the plot (NxyNumCurves).

4.9.1 X-Y plot general parameters

The following parameters represent the arrays of attributes for individual plot curves.

NxyNumCurves	Sets the number of curves.
NxyLineNPoints	Sets the number of points for a single curve.

4.9.2 X-Y plot curve parameters arrays

The following parameters are arrays of size NxyNumCurves. Each curve in a plot has one element from each array that specifies a particular curve attribute. For example, if you were plotting 3 curves, curve 1 would be defined by the attributes in the first element of each of the following arrays, and curves 2 and 3 would be defined by the second and third elements.

NxyLineStyle	Selects whether to use dashed, labeled or automatic line styles.
NxyLineIndex NxyLineThickness NxyLineColor NxyLineSmooth NxyLineDashPattern NxyLineDashLength	Sets the index of the current curve when using the indexed scheme for setting line attributes.
NxyLineLabelText NxyLineLabelFont NxyLineLabelFontColor NxyLineLabelFontSize NxyLineLabelFontThickness NxyLineLabelFontAspect	Sets line label attributes for each curve.

4.9.3 X-Y plot control parameters

These parameters control the display of the plots, such as how to interpret the data with implied coordinates, and how to interpret missing values.

NxyScheme	Specifies whether to interpret attributes as arrays or single values.
NxyControlPlotType	Specifies the input format of the data arrays
NxyYLineN **NxyXLineN**	Where N goes from 1 to NxyNumCurves. These are the X and Y data arrays.
NxyControlRow	Specifies the dimensioning of the X and Y arrays
NxyControlOrder	Specifies the data ordering (C or Fortran) for 2D arrays
NxyControlInvert	Allows user to graph x as a function of y
NxyControlWindow	Allows user to omit curve portions falling outside the grid window
NxyControlMissingValue	Sets the missing values in the data
NxyControlXMin	Specifies the minimum X user coordinate
NxyControlXMax	Specifies the maximum X user coordinate
NxyControlXSmallest	Values less than this value are not considered in the plot calculation if NxyControlXmin is set to NxyControlMissingValue
NxyControlXLargest	Analogous to NxyControlXSmallest
NxyControlYMin **NxyControlYMax** **NxyControlYSmallest** **NxyControlYLargest**	Analogous to the corresponding X parameters

4.10 Contour

The contouring HLU should implement the functionality of Conpack without requiring the user to understand the details of coordinating the Ezmap, Gridal, Areas, Plotchar, and Labelbar utilities. This is a draft list of the resources needed to specify a contour plot. The Contour HLU uses all of the common composite resources. The following resources are specific to creating, filling, and drawing contours.

NcnDataType	Sets whether input data array is short, long, float or double storage type.
NcnDataGridType	The contouring utility will handle regular, regular sparse, irregularly spaced and randomly spaced data.
NcnXDim	Specifies wheter data array's first or second dimension maps to the X dimension
NcnXDataDim	Sets number of elements in X direction.
NcnYDataDim	Sets number of elements in Y direction.
NcnXDataCord	Array of coordinate values of grid points in X direction.
NcnYDataCord	Array of coordinate values of grid points in Y direction
NcnXDataMin **NcnXDataMax**	When grid is regularly spaced these substitute for NcnXDataCord.
NcnYDataMin **NcnYDataMax**	When grid is regularly spaced these substitute for NcnYDataCord.
NcnIntervals	Sets the contour intervals for each contour level.
NcnCIntervalFillPattern	Sets the fill pattern to use for each contour level.
NcnCIntervalColor	Sets the the color index to fill each contour level.
NcnCLineColor **NcnCLineDashPatern** **NcnCLineThickness** **NcnCLineSmooth** **NcnCLineDashLength**	Sets the color index of each contour line.
NcnCLineLabelStyle	Sets contour line label style. Conpack supports 3 ways of labeling contour lines.
NcnCLineLabelText **NcnCLineLabelFont** **NcnCLineLabelFontSize** **NcnCLineLabelFontThickness** **NcnCLineLabelFontAspect** **NcnCLineLabelFontColor**	Strings to label contour lines with.
NcnNumCLines	Number of contour lines.
NcnXStart **NcnXEnd**	Used "clip" X dimension and map a subset of current data into the viewport.
NcnYStart **NcnYEnd**	Used "clip" Y dimension and map a subset of current data into the view port.

NcnInfoLabelText **NcnInfoLabelFont** **NcnInfoLabelFontColor** **NcnInfoLabelFontSize** **NcnInfoLabelFontThickness** **NcnInfoLabelFontAspect** **NcnInfoLabelTextJust** **NcnInfoLabelTextPosX** **NcnInfoLabelTextPosY**	String for informational label.
NcnHighLow	Turns on and of HighLow informational labels.
NcnHighText **NcnHighFont** **NcnHighFontColor** **NcnHighFontSize** **NcnHighFontThickness** **NcnHighFontAspect**	Strings for High informational label. All of the standard text setting resources are implied.
NcnLowText **NcnLowFont** **NcnLowFontColor** **NcnLowFontSize** **NcnLowFontThickness** **NcnLowFontAspect**	Strings for Low informational labels.
NcnMissingValue	Sets a values that represents missing data points in the data set.
NcnHLBox **NcnHLBoxWidth** **NcnHLBoxHeight** **NcnHLBoxLineThickness** **NcnHLBoxLineColor** **NcnHLBoxLineDashPattern**	Turns on High Low Box and sets width and height.

4.11 Vector

There two styles available for displaying vector fields: color encoded and length encoded. The first draws all vectors an equal length, but colors them based upon their magnitude. The second style scales the length of the drawn vector to reflect its magnitude relative to the maximum magnitude.

NvrStyle	Selects between color coded vectors and length encoded vectors
NvrMaxLength	Sets the length of the maximum magnitude vector in output frame coordinates.
NvrIntervals **NvrNumIntervals** **NvrIntervalIndex** **NvrIntervalColor**	The Intervals resource is an array of values used to determine what color a color encoded vector style should use.

NvrLineThickness	Sets thickness of vector line segments.
NvrColor	Used to set the color of vectors when the length encoded vector style is selected.
NvrLabels NvrLabelFont NvrLabelFontSize NvrLabelFontColor NvrLabelFontAngle NvrLabelFontAspect	Writes the value of the magnitude of each vector at the tail of the vector.
NvrLowThreshold	Sets the minumum vector magnitude to be displayed
NvrHighThreshold	Set the maximum vector magnitude to be displayed
NvrDataType	Specifies whether integer or floating point data is being used.
NvrU	Two Dimensional data array for horizontal vector components
NvrV	Two Dimensional data array for vertical vector components.
NvrXDim	Specifies whether first or second dimension of data array maps to the X direction.
NvrXDataDim	Number of element in X dimension of data.
NvrYDataDim	Specifies which dimension of the U/V array maps to the Y dimension.
NvrXDataCoord	Array of coordinate values for X dimension.
NvrYDataCoord	Array of coordinate values for Y dimension
NvrArrowSize	Sets the Size of the arrow drawn on the end of the vectors.
NvrUMissingValue	Sets the value of missing values for U direction.
NvrVMissingValue	Sets the values of missing values for V direction.

4.12 Streamline

Streamlines represent the path taken by particles dropped into a flow field. The following are the resources for configuring a streamline plot style.

NslU	Two dimensional horizontal vector component.
NslV	Two dimensional vertical vector component.
NslColor NslLineThickness	Line attributes for streamlines.
NslDataType	Specifies whether input data is integer or floating point.

NslUMissingValue	Set the value of missing values for the U direction.
NslVMissingValue	Sets the values of missing values for the V direction.
NslXDataCoord	Array of coordinate values for the X dimension.
NslYDataCoord	Array of coordinate values for the Y dimension.
NslXDim	Secifies whether first or second dimension of data array maps to the X direction.
NslXDataDim	Number of elements in X direction of data.
NslYDataDim	Number of elements in Y direction of data.
NslArrowSize	Size of arrows drawn on streamline.
NslSpacing	Controls spacing between streamlines.
NslStartEligible	Controls which grid boxes are canidates to start a streamline.
NslStepsPerGridBox	Controls number of sample points per grid box that a streamline passes through.

4.13 Common 3-D resources

The following resources are common to both the Isosurface and Surface utilities.

N3dEyePoint	coordinate postion of the eye
N3dViewCenter	Coordinates of position looked at
N3dConstLines	Sets the constant lines (U, U and V, U and V and W, etc.) to be used for drawing a surface
N3dSkirt	Turns off/on a wall or skirt along the edges of a surface where it intersects the data boundary
N3dContourHigh **N3dContourLow** **N3dContourInterval**	Sets the high, low and interval for drawing contours on a surface

4.14 Surface

Surface plots are used to create a three-dimensional perspective of two-dimensional data with hidden lines removed. The surface, Z, is a function of the two variables X and Y.

For example, a three-dimensional surface could be created from a two-dimensional array of data where the first array dimension is the X position and the second dimension is the Y position on a Cartesian grid. The value (Z) of each element in the array defines the height of surface for each X and Y coordinate.

The following parameters define the characteristics of the surface such as rotation, surface representation, and viewing angle.

NsrArrayDims	Sets the X and Y data dimensions
NsrArray	The data array
NsrArrayRange	Specifies how much of the data set to use or subsample
NsrArrayXstride **NsrArrayYstride**	Sets the stride factor by which the utility will sample the data
NsrXcoord **NsrYcoord**	Arrays of X and Y coordinates, respectively
NsrEyepoint	Position of the viewers eye
NsrViewCenter	Position the viewer's eye looks at
NsrStereo	Sets whether or not to draw stereo images
NsrStereoAngle	Sets the relative angle between the eyes for stereo images
NsrStereoType	Sets how stereo images are placed on frames
NsrPlotDirection	Sets which plotting direction corresponds to the positive Z axis
NsrConstLines	Sets which lines to use to draw a surface
NsrDrSide	Sets which side of a surface to draw
NsrSkirt	Specifies whether or not to draw a skirt around the object
NsrSkirtBottom	Sets the level at which the bottom of a skirt terminates
NsrNLevels	Sets the number of levels of constant Z
NsrStereoTheta	Sets the angle in radians between eyes for stereo pairs
NsrContourHigh **NsrContourLow** **NsrContourIncrement**	Sets the highest and lowest levels of constant Z and the increment between them
NsrSpvalFlag	Controls the use of the special value (NsrSpval)
NsrSpval	The real data value used to mark special or missing data

4.15 Isosurface

This section describes the resources that specify an isosurface object. These objects are created by taking a 3D volume of data and specifying a threshold value. A polygon surface is created that intersects values in the data which equal the threshold value.

These surfaces or objects can then be operated upon. For example, a user can change the object's color, position, or lighting characteristics.

Besides iso-surfaces, this section describes the resources for 3D surfaces generated from 2D data (the Z position is determined by the data value) and volumentric objects (data points are displayed as relative intensities where the intensity is proportional to the data value).

The 3D functionality is divided into two utilities: NCAR Graphics and PolyPaint.

The NCAR Graphics utility will be built on the NCAR Graphics ISOSRF routine which generates isosurfaces from a three-dimensional array.

The need to develop a programming interface to the PolyPaint application has not been confirmed. The interface presented here for PolyPaint is a possiblility of what the interface may look like should it be determined that this is a high priority.

The PolyPaint utility will be built on MMM's PolyPaint application which also generates isosurfaces, but it includes support for color shading, volumentric rendering, and index and true color. PolyPaint also allows the user to control lighting, viewing, and shading.

The resources for 3D surfaces are divided into three sections: NCAR Graphics isosurface resources, common resources between the NCAR Graphics Isosurface utility and PolyPaint, and PolyPaint resources.

NisConstLines	Specifies which type of iso-surface lines to draw (U,U and V, U and W, U and V and W, etc.)
NisVisibility	Sets what data is inside and outside a surface
NisEyepoint	Position of the eye in 3-space
NisArrayDims	The X, Y, and Z dimensions of the data
NisArrayRange	Specifies the subset of the data to use
NisArrayXstride **NisArrayYstride** **NisArrayZstride**	Specifies how much, if any, data to thin from the data set
NisAxes	Flag for turning on/off the coordinate axes
NisThreshold	Value which defines the iso-surface
NisSurfaceData	An array of data from which an iso-surface is computed.
NisAliasExpon	Sets the alias exponent (depends on the monitor used to display antialiased wire-frames)
NisAmbient	Sets the ambient light level intensity
NisXScale **NisYScale** **NisZScale**	Stretching factors for each axis. This allows the user to stretch and shrink each dimension.
NisObjectBackIntensity	Sets an object's intensity as seen through a transparent object
NisObjectFrontIntensity	Sets an object's transparency intensity
NisPerpAxis	Sets the axis which is perpendicular to the base plane
NisBasePlaneLevel	Sets the level of the base plane
NisRedLimitMin **NisRedLimitMax** **NisGreenLimitMin** **NisGreenLimitMax** **NisBlueLimitMin** **NisBlueLimitMax**	Sets the range of brightness associated with the red, green, and blue raster-mapping arrays
NisViewCenter	Sets the center of the object. The imaginary eye looks at this point.
NisPartitionNumber	Sets the number of color partitions a color table should be divided into

NisColorSet	Sets the current color partition or the surface color of the next object if using true color. Why not use NgmColorMap resource????????
NisCoordTrans	Specifies the coordinate transformation
NisCrossSectColorHigh **NisCrossSectColorLow** **NisCrossSectColorInterval**	Resources for cross section coloring
NisCrossPartition	Sets the color partition to use for index color
NisVolCutPlane	Sets the level whereby values greater than the level are displayed if NisVolHigher is on, and values less than the level otherwise.
NisVolHigher	Toggle flag used to determine what values of a surface to display
NisCuttingPlane	Sets the cutting plane level in a data volume
NisAxisCutPlane	Sets the axis which is perpendicular to the cutting plane
NisShadowDarkness	Sets the darkness of a shadow which falls on an object
NisSpecularReflectionExponent	Sets the brightness of specular highlights
NisSpecularIntensity	Sets the intensity of specular highlights if using indexed color or the color if using true color
NisThinSurfaceTransparency	Sets the value of the exponent for thin surface transparency
NisTransTable	Table for determining which objects are transparent to which other objects
NisGeoTransformation	Sets the coordinate transformation to use for geographic data
NisHazeColor	Sets the haze color (true color) or the brightness (indexed color) which an image fades to depending on distance from eye
NisInnerTransExponent **NisOuterTransExponent**	Define the inner and out exponents used for transparency level in nested shells.
NisLightNumber	Sets the number of lights in the scene
NisLightIntensity	Sets the intensity for diffuse lighting of the current light source for index color or the color of the current light source for true color.
NisLightIndex	Sets the current light source
NisLightPosition	Sets the coordinates of the current light
NisMagnification	Sets the amount of object magnification
NisViewAngle	Sets the viewing angle for enlarging and shrinking an object
NisColorMidPoint	Sets the midpoint value for the current color partition. This is used for gamma correction.
NisObjectIndex	Sets the current object

NisPartitionIndex	Sets the current color partition
NisTransOverlapColor	Sets the color map to use for index color when pixels are overlapped by a transparent object.
NisRampNumber	Sets the number of color ramps for true color
NisRampColor	Defines a color ramp for true color applications
NisRampId	Sets the current data color-coding ramp
NisRotationAngle	Sets the object rotation
NisShadowPosition	Sets the Z position of a shodow cast onto a plane
NisShellLimitLow NisShellLimitHigh NisShellLimitInterval	Define the surface and intervals of shells
NisVerticeLimit	Sets the maximum number of vertices a polygon can have
NisObjectActive	Toggle parameter which determines if an object is active and will be rendered
NisDataAlternate	Determines whether to use color or main data for cutting plane
NisPixelAlternate	Sets how to handle transparency (alternating pixels)
NisAntiAlias	Toggle for turning antialiasing on and off
NisArrayScale	Toggles between scaling and not scaling the cutting-plane coloring by the values in the cutting-plane array
NisAutoHaze	Toggles the auto haze calculation
NisAutoCenter	Determines how to set the center of view
NisAutoScale	Determines how the view angle is set
NisSurfaceNormals	Toggles between smooth and faceted surface
NisBackfaceCull	Toggles backface culling
NisScaleBySlice	Selects option to scale cross-section coloring by values in an array slice
NisColorCoding	Sets whether or not to create and use color-coding data when creating polygons
NisSurfaceColorData	An array of data which is mapped onto an iso-surface.
NisShellTransparency	Sets whether or not to make transparent shells different colors
NisLightType	Sets the current light's type (directional/point)
NisEyeRotation	Toggles ability to rotate the eye position
NisLightRotation	Toggles ability to rotate the light position
NisShellLow	Determines wheter low or high values are to be on inside of a surface

NisShading	Sets the shading algorithm (phong/gouraud)
NisSmallOpaque	Allows user to make smallest shell in nested shells opaque
NisShadowProjection	Sets how an object's shadow is projected
NisCuttingPlaneColor	Toggles between coloring and not coloring the interactive cutting plane
NisSidewallPlanks	Sets how the side walls are calculated
NisRasterScale	Sets how to scale the red, green, and blue raster arrays for true color
NisRasterMap	Sets whether or not to superimpose raster data onto the surface of an object (true color)
NisSelfTransparency	Sets whether an object should be transparent to itself (index color)
NisWireShades	Sets whether or not to shade wire frames according to current lighting model
NisSurfaceFaceUp	Determines whether or not surfaes face up or down surrounding higher and lower values
NisLightTerminator	Sets the angle for light termination
NisThinSurface	Toggle for using thin surface transparency
NisTrueColor	Sets whether to use index or true color
NisVolhigher	Sets whether to display higher or lower values than NisVolCutPlane
NisVolAlgorithm	Selects the volumetric algorithm (quick/slow)
NisSkirt	Sets whether or not to build walls or endcaps
NisWireFrame	Selects between wire frame or surface rendering
NisBoundingBox	Sets whether or not to display a bounding box around the data
NisZoomFactor	Sets how much to zoom in or out on an object
NisImageType	Sets how to render an image
Nis2DAxis	Sets the axis perpendicular to the plane in which 2D contours or surfaces are generated
Nis2DContourLow **Nis2DContourHigh** **Nis2DContourInterval**	Set limits for 2D contours
Nis2DWidth	Sets the width of 2D contours
Nis2DLevel	Sets the level in a data volume at which 2D contours or surfaces are generated
Nis2DPlotLevel	Sets the level at which a 2D surface will be plotted.
Nis2DSurfaceScale	Sets the amount which data values are scaled in a 2D surface plot

4.16 Histogram

@@@To be added.

4.17 Annotation

The Annotation utility should allow the user to add text, lines, and shapes to any CGM frame. Users should be able to develop custom lines and fill patterns. The annotation utility will also be able to draw label bars and legends to meet customized demands.

SECTION 5

NCAR Command Language Specification

5.1 Introduction to NCL

The NCAR Command Language (NCL) will support the needs of users who want to conduct interactive data exploration and analysis. NCL uses the following model for interactive data analysis: data are selected and read from a file, data are processed, a visualization specification is made, then the visualization is rendered. To provide true interaction, NCL must allow the user to return to any of the previous steps in this model at any time.

Easy and intuitive access to datasets is a fundamental prerequisite for interactive data exploration, analysis, and visualization. Since datasets often come in a variety of data formats, grid sizes, grid resolution, and units, very different datasets often need to be combined, compared, and used at the same time. Currently, specialized applications must be developed to read individual datasets and transform them into a form that is compatible with other datasets being used, as well as with the graphics package being used.

NCL allows different datasets to be imported into one uniform and consistent data manipulation environment. The primary data format used by NCL is the netCDF data format. This network-and-architecture-transparent format has the ability to store multiple data types of multiple dimensions. Furthermore, the netCDF file contains information describing its content. For example, a given variable can have information about the units it is stored in, missing values, valid data ranges, etc. This general information facilitates the development of a powerful generalized data manipulation environment like NCL. For

more information on netCDF, consult the *netCDF User's Guide*. For a good understanding of NCL, you should be familiar with netCDF and basic programming language concepts.

Easy and intuitive output specification is also a prerequisite for interactive data analysis and exploration. NCL has built-in defaults for quick and accurate visualization of data. The defaults can also be customized by the user in the form of a user defaults file. Furthermore, visualization specifications can be entered and altered at the command line.

This section of the NCAR Interactive functional specification is organized into four sections. The first section presents the NCL syntax and describes the semantics of the language. The second section describes the visualization specification process. The third section outlines the set of built-in data manipulation functions provided with NCL. The fourth section describes the requirements for user extension of the NCL function set.

5.2 Language overview

NCL can be thought of as a complete programming language. It has types, variables, operators, expressions, conditional statements, and loops. NCL also has features that are not found in common programming languages. These additional features handle manipulation of metadata and the configuration of the output graphics. NCL can operate in three modes. The first mode of operation is as an interactive command line interpreter where every statement is executed immediately after the user enters a command or expression. The second mode of operation is as a batch command interpreter where an entire NCL script is read in at one time and executed. The third mode is intended to facilitate the batch production of large quantities of output visualizations either for data exploration or video production. NCL is not meant to be a replacement for

programming in other structured languages, but it is meant to provide an integrated environment where data can be selected, manipulated and visualized interactively without requiring compilation.

5.2.1 NCL data model

As explained earlier, one dominant problem in handling scientific data is handling the data's metadata, the information that describes the data. Some common metadata items are units, dimensionality, variable names, valid ranges, data minimums and maximums, missing values, coordinate indices, and brief textual descriptions. Much of this information is extremely useful for creating informative visualizations of the data. The goal of NCL's data handling capability is to manage the metadata in a convenient and straightforward fashion. The data model used in the netCDF datafile format is an ideal candidate for NCL's internal data representation. NetCDF provides a common interface for storing multi-dimensional data and its associated metadata.

The primary model of data in NCL is the netCDF data file model. This does not mean that the only format supported by NCL is netCDF, but simply that the abstraction NCL uses to represent data is the same. Most other data formats can be mapped into the netCDF model and therefore used in NCL. NCL data is divided into files that each contain named variables, named dimensions, and attributes. The attributes are file attributes that describe the file; they can provide a title, a file history, etc. The variables are multidimensional arrays with each dimension named. Each variable can have attributes as well. These are attributes like the valid range of the variable, the units, the long name of the variable and other descriptive information. A data file can have as many variables and attributes as the user desires.

Another property of the NCL data model is the concept of the coordinate variable. A coordinate variable, by definition, has

the same name as its dimension. It holds the coordinates for that dimension. For example, a coordinate variable for the dimension *lat* would hold the coordinates in degrees latitude of each integer index. All of these features make up the NCL data model. Special syntactic constructs have been included in NCL to handle some of these features.

file1@title = "Temperature"	creates file attribute title and assigns it the string "Temperature"
file1.a = var1	assigns var1 to file1 and renames it a.
file1.a@units = "Degrees C"	creates variable attribute units and assigns it the string "Degrees C".
file1!0 = "lat"	renames the first dimension of file1to "lat".
file1&lat = [90,45,0,-45,-90]	creates the coordinate variable for the dimension called lat if file1.

5.2.2 NCL data types

In NCL, there is only one abstract data type, the *file* record. The file record is similar to the record type in pascal, the structure in C, and the common block in FORTRAN. The NCL file record holds variables, dimension names, variable and file attributes, missing value values, etc. A file record can hold any number of these items with any number of primitive data types. The NCL record is called a file record because everything held by a file record can be written into one netCDF file. When ascii or binary files are created from file records, a separate descriptor file is created in which all of the metadata for the variables in the file record are stored. The descriptor file also describes the organization of the data in the binary or ascii file. Similar procedures are used to map the file record to other storage formats. The NCL file record allows the user to group variables and other related information logically and conveniently.

The other items like the attributes, the coordinate variables, and dimensions should be thought of as fields of a record and not types of their own. These are all elements of the data model NCL will use. Variables are all array type variables, with scalars being arrays of one dimension of size one. Variables, attributes, and coordinate variables can be one of the following primary data types: *float*, *double*, *short*, *long*, *character*, or *byte* scalar value or multi-dimensional array. Dimension names are of type character. File types have no primary type because they are just a symbolic representation of the file. The only operations that can be performed on a file type are assignment, passing file record variables as parameters to functions, and adding and deleting variables, attributes and dimensions from file records.

Variables can exist without being associated with a file record. These variables are called variables in memory. They can be assigned to a file, as the examples in the previous section show.

5.2.3 Variables and data selection

NCL variables are, in many ways, just like variables in other programming languages; they are symbolic representations of either scalar values or arrays. In NCL, variables must begin with a letter and can have any combination of characters and numbers making up the rest of the name. The following is an example of an instantiation of an NCL variable. The prompt "ncl>" represents the command line prompt provided by the NCL interpreter.

```
ncl> var_1 = 5
```

As expected, the scalar integer 5 is assigned to the variable "var_1" in memory. To instantiate a variable and associate it with a file record, the filename is prepended to the variable name. For example:

```
ncl> file1.var_1 = 5
```

Now the value 5 is assigned to the variable "var_1" in the file "file1". To add another variable to the file record "file1", the new variable name is appended to the file record name. For example:

```
ncl> file1.var_2 = 5.0
```

Now the scalar float value 5.0 is assigned to the variable "var_2" in the file "file1". An array of values can also be assigned to a variable. For example, the following three examples create a one, two, and three-dimensional array respectively:

```
ncl> a = [ 1, 2, 3 ]
ncl> b = [ 1, 2, 3; 4, 5, 6 ]
ncl> c = [ [ 1, 2, 3; 4, 5, 6 ] ; [ 7, 8, 9; 10, 11, 12] ]
```

Array "a" is a one-dimensional array of integers of size three in memory. Array "b" is a 2x3 two-dimensional array of integers in memory, and array "c" is a 2x2x3 three-dimensional array of integers in memory. In the above examples, each group of consecutive numbers separated by commas is an individual row of the array. Rows are separated by semicolons. Therefore array b is a 2x3 array because there are two rows with three elements each.

5.2.3.1 Standard array indexing

NCL provides three basic types of array indexing, the first of which is discussed here and referred to as standard array indexing. The array indexing presented here is in many ways similar to how standard programming languages implement array indexing. The other types are specific to NCL and provide the user different ways to work with and select their data. The reasons for having three different types of array indexing is to provide alternatives to the user that offer an easier and more intuitive way to select data. NCL arrays are ordered using "row by column" ordering. NCL's standard array indexing is similar to FX/8 FORTRAN. This means that both individual elements and ranges can be indexed. Array indexes

begin at zero and end at n-1 where n is the size of the array. The following are examples of NCL's array indexing.

```
Example 1)
ncl> print(a)
(0) 1
(1) 2
(2) 3

Example 2)
ncl> print(b)
(0,0) 1
(0,1) 2
(0,2) 3
(1,0) 4
(1,1) 5
(1,2) 6

Example 3)
ncl> print(c)
(0,0,0) 1
(0,0,1) 2
(0,0,2) 3
(0,1,0) 4
(0,1,1) 5
(0,1,2) 6
(1,0,0) 7
(1,0,1) 8
(1,0,2) 9
(1,1,0) 10
(1,1,1) 11
(1,1,2) 12
```

Examples 1, 2, and 3 show references to the entire arrays. When a variable is referenced by name without any indices, the entire array is referenced. The print command can be thought of as a procedure in NCL and the variables are passed to the procedure as parameters. Additional information about procedures appears later in this document. Also shown here are the indices of each of the elements in the array. This should be helpful for understanding the rest of the examples.

```
Example 4)
ncl> print(a[1])
(0) 2
```

```
Example 5)
ncl> print(b[0][2])
(0) 3

Example 6)
ncl> print(c[1][1][1])
(0) 11
```

Examples 4 through 6 show how individual elements of each of the example arrays can be indexed. Since variable "a" is one-dimensional, only one index is needed. Similarly for the three-dimensional array "c", three indices are needed to select one element from the array.

```
Example 7)
ncl> print(b[1])
(0) 4
(1) 5
(2) 6
```

Example 7 demonstrates how arrays are indexed when the number of subscripts is less than the number of dimensions. NCL uses the rule that when there is no index for a particular dimension, the entire dimension is selected. As shown, example 7 indexes the entire second row of "b".

```
Example 8)
ncl> print(a[0:1])
(0) 1
(1) 2

Example 9)
ncl> print(a[1:*])
(0) 2
(1) 3

Example 10)
ncl> print(a[*:1])
(0) 1
(1) 2
```

Examples 8 through 10 demonstrate various ways in which ranges of dimensions can be specified. Example 8 shows how a start and end index is specified. In examples 9 and 10, the "*" means to select everything from either the start index to the

end as in example 9, or to select everything from the beginning to the end index as in example 10.

```
Example 11)
ncl> print(c[0][*][0:1])
(0,0) 1
(0,1) 2
(1,0) 4
(1,1) 5
```

Example 11 shows all three indexing methods. In the first dimension, only the first row is selected. In the second dimension, everything is selected and in the third dimension, the first two elements are selected. Since there are only two dimensions with more than one element selected, the resulting array is a two-dimensional array.

5.2.3.2 NCL specific array indexing

Additional details about the NCL data model are needed before discussing NCL-specific array indexing. As mentioned earlier, the NCL data model is a virtual representation of the netCDF model of scientific data. One important netCDF concept is the concept of the coordinate variable and the ability to name variable dimensions (see the *NetCDF User's Guide*). NetCDF provides a mechanism for naming dimensions.

For example, if a data file contains one independent variable, such as temperature, and if temperature is defined over latitude longitude, altitude, and time, netCDF allows the user to name each of these dimensions. Therefore, one possible naming scheme is to call the dimensions, "lat," "lon," "level," and "time." It is most likely apparent to the owner of this data that latitude and longitude are coordinates in degrees, level is in millibars, and time is in hours. These are coordinates, and for someone familiar with their data, using these coordinates could provide a more convenient way to index their data than the previously described integer indexing scheme.

The netCDF file defines a coordinate variable to be a variable that has the same name as a dimension. This variable holds the coordinates of each element in the data array. The following example NCL script is provided for clarification. (This example shows some functions that have not yet been discussed in this document).

```
ncl> addfile("T.cdf","NETCDF")
ncl> inquire
FILE        VARIABLE   DIMENSION      DIM NAMES
--------------------------------------------------------------------------
T.cdf       temp       8x10x33x36     [frtime]x[level]x[lat]x[lon]
T.cdf       lat        33             [lat]
T.cdf       lon        36             [lon]
T.cdf       level      10             [level]
T.cdf       frtime     8              [frtime]
ncl> print(T&level)
FileRead reading 40 bytes. Please wait...read successful
(0) 1000.000000
(1) 850.000000
(2) 700.000000
(3) 500.000000
(4) 400.000000
(5) 300.000000
(6) 250.000000
(7) 200.000000
(8) 150.000000
(9) 100.000000
ncl> a = T.temp[ * ][ 3 ]
```

This example loads the netCDF file "T" into NCL. Then the inquire procedure prints out the currently defined variables. This file contains five variables, four of which are coordinate variables by definition. The print procedure prints out the values of the "level" coordinate variable. The resulting printout shows how the integer indexes map to the coordinate values.

Now for the sake of example, suppose that the user wants to select the temperature at every latitude, longitude, and forecast time, but only wants to look at values that correspond to the 500mb pressure level. With only standard integer indexing available, the user would have to figure out which integer index corresponds to the correct pressure level. Certainly

printing out the coordinate variable level provides this information, but this is an extra unneeded step.

The solution to this problem is the second type of NCL array indexing, called coordinate indexing. The following is an example of this type of indexing:

```
ncl> a = T.temp[ * ]{ 500 }
```

Essentially this performs the same selection as shown in the previous example. The only difference is that "curly" style brackets (braces) are used in place of the square brackets. The use of the braces instructs NCL to choose the integer index based on the coordinate value entered.

In this example, NCL finds the integer index, 3, for the coordinate value 500 in the "level" coordinate variable. There is a simple rule for choosing the index when the coordinate given is not an exact coordinate value. NCL finds the coordinate value above and below the given value and chooses the closest one. This is only the case when one coordinate value is given. If a coordinate value range is specified, then all coordinate values within the range are selected. As with integer indexing, a range of values can be specified:

```
ncl> a = T.temp[ * ]{ 700 : 200 }
ncl> b = T.temp[ * ][ 2 : 7 ]
```

The above examples are equivalent selections. Providing the coordinates values allows a more intuitive understanding of the part of the data being selected. The second example uses the integer indices to perform the selection. This method really doesn't convey any useful information about the selection operation on the data.

This second type of indexing scheme allows users to take advantage of the coordinate variable feature of the netCDF model to select data by the coordinate it is defined in, rather than the integer coordinates the array is defined in. This indexing scheme is called NCL coordinate variable indexing.

The third and final indexing type is a variation on the previously mentioned indexing schemes. In both the previous schemes, the variable subscripts had to be listed in the order in which they are defined in the netCDF file. This final scheme allows the subscripts to be listed in any order. In some cases this may be the most intuitive way to select data from a NCL file record or memory variable. The following is the same selection as the previous example using the third type of indexing, which is called named indexing.

```
ncl> a = T.temp{ level | 700 : 200 }
ncl> b = T.temp[ level | 2 : 7 ]
```

Named indexing allows the user to use the dimension name to specify the dimension that the subscript indexes. The rule for braces is the same as the preceding indexing schemes. The difference is that in each subscript, the name of the dimension is listed followed by the pipe symbol (|). All dimensions that are not subscripted are selected over their entire ranges by default.

5.2.3.3 Accessing and Assigning Metadata Values

Several language constructs have been introduced in the NCL language to handle the management of metadata like dimension names, coordinate variables and variable attributes.

The operator '!' is used to select dimensions. There are two possible uses of this operator. First, '!' can be used to retrieve dimension names for variables or files. The following are examples of this use.

Example 1)

```
ncl> print(file1.var!0)
(0)"lat"
ncl> print(file1.var!1)
(0)"lon"
ncl> print(file1.var!2)
Error: file1.var only has two dimensions!
```

Example 2)

```
ncl> file1.var!0 = "latitude"
```

Example 3)

```
ncl> print(file1.var!0)
(0)"lattitude"
```

Example 4)

```
ncl> print( [file1!0 , file1!1 , file1!2 )
(0) "time"
(1) "lon"
(2) "latitude"
ncl> print( [ file1.var!0, file1.var!1 ])
(0) "latitude"
(1) "lon"
```

Example 1 above prints out the dimensions of the variable *var* in the file *file1*. Example 2 renames the first dimension of variable *var* to *latitude*. Example 3 shows the changed dimension name. Finally, example 3 shows how the file dimension name has been updated to reflect the name change. When the '!' operator is followed by an integer value, the integer represents the number of the dimension to access. Note the difference between the first dimension of the variable *var* and the first dimension of the file *file1*. Variables that are members of a file record use dimensions defined in the file, but they do not necessarily use the dimensions in the same order as they are defined in the file. In this example variable *var* is a two dimensional array of lat by lon in row/column order. The ordering of the dimensions in the variable *var* reflect the ordering of the data. The ordering of the dimensions in the file *file1* simply represent the order in which the dimensions were defined. In any event , changing the name of a dimension either at the file level or the variable level causes the dimension names of all variables using that dimension to be changed.

Dimension sizes can not be changed ever. Dimension sizes can be defined using the built-in function **varcreate**, which creates a new variable with dimension and size information as input. Dimension size information can be listed using the built-in

function **dimsize**, which returns the size given the file or variable name and the dimension name.

The second use of the '!' operator is to place the dimension name after the operator. Example 2 would appear as follows:

ncl> file1.var!lat = "latitude"

This redefines the dimension name *lat* to *latitude* just as example two did.

Another construct added to the NCL is the '&' operator which allows users to associate a coordinate variable with a dimension. The following example associates an array of coordinate values with the dimension named *latitude*.

```
ncl> file1.var&lattitude = [ -90.0, -80.0, -70.0 . . . 90.0]
```

The operator '&' must be followed by a valid dimension name for the file or variable appearing before it. Furthermore the dimension size of the coordinate array must be the same as the valid dimension. If these two conditions are met the array becomes the coordinate array for every variable in the file containing the dimension. The following ncl statement accomplishes the exact same operation as the above statement since dimensions and coordinate variables are global to the file record.

```
ncl> file1&lattitude = [-90.0, -80.0, -70.0 . . . 90.0]
```

The final construct introduced in ncl for accessing metadata is the '@' operator. It is used to assign attributes to variables and file. Its syntax is very similar to the '!' and '&' operators.

```
ncl> file1@title = "This is the main title attribute"
```

The above assigns the string "This is the main title attribute" to the file attribute *title*. If the attribute title was not previously defined, it is added to the files attribute list. If the attribute was defined the old title is over written with the new one. What ever the string following '@' is, is the name of the attribute. If

a variable instead of a file precedes the '@' then the attribute is a variable attribute. For example:

```
ncl> file1.var@units = "Degrees C"
ncl> print(file.var@units)
(0) Degrees C
```

The above example creates the attribute units and assigns the string "Degrees C." The print command shows how to access attributes.

5.2.4 NCL expressions and operators

NCL expressions are quite similar to expressions in traditional programming languages. By definition, an expression is anything that returns a value and a type. References to variables and constants are expressions. Subscripts are also expressions, as are functions and parameters to functions.

The following are all examples of expressions:

```
a[0]
2.0
a * 2.0
"Hello World"
a!0
a&lat
```

There are 20 operators in the NCL language. Table 1 lists all 20 operators. These operators are separated into hierarchical precedence groups. The operators at the top of the list are the highest precedence, and the operators at the bottom are the lowest precedence. When operators are members of the same precedence group, a left-to-right rule is used to evaluate the expression. For example, A*B/C would evaluate A*B first and then divide the result by C. A-B/C would evaluate B/C first (because division is in a higher precedence group than subtraction), then subtract the result from A. This is consistent with standard programming languages. To force a particular order of execution left and right parentheses are used. (A-B)/C, as expected, would compute A-B and divide the result by C.

TABLE 8 NCL Operators

Symbol	Operation	Class
-	Unary Negative	Unary
+	Unary Positive (Absolute Value)	Unary
NOT	Logical Not	Unary
^	Exponentiation	Algebraic
*	Multiplication	Algebraic
/	Division	Algebraic
#	Matrix Multiplication	Algebraic
%	Modulo	Algebraic
+	Addition	Algebraic
-	Subtraction	Algebraic
<	Less Than Selection	Algebraic
>	Greater Than Selection	Algebraic
LE	Less Than or Equal To	Relational
LT	Less Than	Relational
GT	Greater Than	Relational
GE	Greater Than or Equal To	Relational
NE	Not Equal To	Relational
EQ	Equal To	Relational
AND	Logical And	Relational
OR	Logical Or	Relational

The operators can be divided into three classes: *Unary*, *Relational* and *Algebraic*. *Unary* operators require only one operand and produce either a variable data type or a Boolean data type as a result. *Boolean* types were not included in the data type section because there is no netCDF representation of a Boolean variable type. However, an expression can yield a Boolean type in an expression where *Relational* operators compare two operands. The *Algebraic* operator type takes two operands and produces a result type that can be assigned to an

NCL variable. Table 2 outlines result types for each operator based on operand data types.

TABLE 9 NCL Operator result types

Symbol	Left Operand Type	Right Operand Type	Result Type
=	Numeric Type Character File	Numeric Type Character File	n/a
^	Numeric Type	Numeric Type	Numeric Type
*	Numeric Type	Numeric Type	Numeric Type
/	Numeric Type Numeric Type	Numeric Type Integer	Numeric Type Integer
%	Integer Only	Integer Only	Integer
#	Numeric Type	Numeric Type	Numeric Type
+	Numeric Type	Numeric Type	Numeric Type
-	Numeric Type	Numeric Type	Numeric Type
<	Numeric Type	Numeric Type	Numeric Type
>	Numeric Type	Numeric Type	Numeric Type
LE	Numeric Type	Numeric Type	Boolean
LT	Numeric Type	Numeric Type	Boolean
GE	Numeric Type	Numeric Type	Boolean
GT	Numeric Type	Numeric Type	Boolean
EQ	Numeric Type	Numeric Type	Boolean
NE	Numeric Type	Numeric Type	Boolean
NOT	n/a	Boolean	Boolean
-	n/a	Numeric Type	Boolean
+	n/a	Numeric Type	Boolean
OR	Boolean	Boolean	Boolean
AND	Boolean	Boolean	Boolean

Note that no operators operate on file types other than the assignment operator. When operand types are not identical, NCL attempts to implicitly coerce the operands to be matching

types. Table 3 shows the types can be coerced and what types they can be coerced to.

TABLE 10 NCL Coercion Table

Type	Coercible To
Short	Long Float Double
Long	Float Double
Float	Double

Any combinations not appearing in this table will result in error messages. There is one exception to the coercion rule. When the right operand of the division operator is an integer, the operand is not coerced because integer division is used.

In NCL, operands can be multidimensional; if *a* and *b* are arrays of equal dimensions and dimension sizes, then *a* * *b* multiples every element in *a* by the corresponding element in *b*. Similarly, if *a* is an array and *b* is a scalar, then *a* * *b* multiplies every element in *a* by the value of *b*. Operands of an algebraic and relational operator must have identical dimensions, or one of the operands must have one dimension of size one. Relational operators work in the same fashion. The conditional expression *a* LE *b* is only true when every element in *a* is less than its corresponding value in *b*. The only exception to this rule is for the # operator, which requires two dimensional arrays where *a*'s dimension sizes are *m* by *n* and *b*'s are *n* by *m*.

NCL uses "lazy" evaluation to evaluate relational expression. This means that as soon as enough terms in a Boolean expression have been evaluated to evaluate the complete expression, the expression is evaluated. Consider the following expression:

(a LE b) and (c GT d) or (m NE 0)

If and only if the first term, a LE b, evaluates to true does the second term, c GT d, get evaluated. Similarly, if and only if the first two terms evaluate to false does the third term get evaluated. This is what is meant by "lazy" evaluation.

5.2.4.1 Expressions Containing Missing Values

There is one attribute that NCL absolutely depends on. The name of this attribute is "_FillValue." This attribute is a number of the same data type as its variable that fills data points where the values at that point are missing. NCL uses this attribute to exclude any data points containing this value from expressions. Excluding missing values from being used in expressions is very computationally expensive but also important. Special effort will be made in the design to focus on this problem. When missing values occur in expressions the result of the expression will have a default missing value set at every point where a missing value occurred in the variables used to compute the expression. Most likely there will be a commandline option for turning this feature on and off. When turned off the most efficient expression evaluation will be used. Deleting the "_FillValue" attribute has the same effect. This must be done if attempting to fill in missing values otherwise the missing values are excluded from the expression that's trying to fill them.

5.2.4.2 Expressions and Metadata

The handling of metadata in expressions presents some problems. Consider the following example where file records *east* and *north* contain vector variables *U* and *V* respectively. The variable c ends up being the total wind speed at every point in the variables *U* and *V.*

c = sqrt(east.U^2 + north.V^2)

If *east.U* has an attribute called *longname* which is set to "eastward wind component" and *north.V* has also has an attribute called *longname* which is set to "northward wind component." Obviously neither of these attributes is applicable to the new variable *c*. Similarly the attributes *valid_range*, *valid_min*, *valid_max* and *scale_factor* are not valid for the variable *c*. There is also no guarantee or check, when computing an expression, that the coordinate variables and the dimension names are the same between all variable in an expression. The only constraint placed on variable in an expression is that the dimension sizes be the same.

The solution to his problem is when two or more variables are used in an expression the coordinate values and dimension names from the first variable in the expression are used and attribute are not propagated in to the result. If there is only one variable in an expression then just the coordinate variable and dimension name information is propagated. In either case all attribute information is not propagated except the "_FillValue" attribute which is a special reserved attribute. If the variables making up the expression contain more than one "_FillValue" value then the default "_FillValue" value for the result type is used to fill in the missing values in the result. This is not an ideal solution but is the only one that can be applied universally to all expressions and variables.

If the user wish to use the coordinate variable and dimension names from a variable, other than the first one in the expression, they must use another operator that would allow them to choose from which variable in an expression to copy dimension name and coordinate variable information into the result. Attributes would still not be copied because more often than not attributes change in expressions. An example of an operator to do this kind of selection follows.

```
ncl> a = file1.var - file2.var'
ncl> print( a!0)
(0) "lat"
ncl> print(file2.var!0)
```

```
(0) "lat"
ncl> print(file1.var!0)
(0) "latitude"
```

5.2.5 Functions and procedures

In NCL, the difference between functions and procedures is that functions return a value and procedures don't. Functions are expressions and can therefore be used in an expression. The syntax of function and procedure calls is the same throughout. The name of the function or procedure is followed by a parameter list. A parameter list has individual arguments separated by commas and enclosed in parentheses.

```
function(param1,param2,param2)
```

Parameters in NCL are passed by reference only. This means that a function and procedure can change the value and have that change reflected in the variable after the function or procedure has ended. Below are examples of calling functions and procedures.

```
ncl> writefile("/u1/ethan/mydata.cdf",file1)
ncl> a = sin(file1.a)
```

The first example is a procedure that writes the file record *file1*. The second is a function that returns the sin of the variable *file1.a*.

The syntax for defining NCL functions and procedures requires the keywords *function* or *procedure*, followed by the name of the function or procedure, which is followed by a parameter declarations list. The body of the function or procedure needs to be enclosed by a *begin* and an *end*. The following are examples of function and procedure definitions.

```
function sqrt(x) begin
        sqrt = x ^ .5
end

procedure sqrt(x,y) begin
```

```
        y = x ^ .5
end
```

The first example defines a function called *sqrt*. For functions, the function name is used to assign the return value. The second example shows how a procedure can be used to accomplish the same task. In both examples, a very simplified parameter declarations list is used. The format of the parameter declaration list can be very complex. In the two examples, the most general format was used.

When the parameter list only contains the names of the parameters, any type and size of variable can be passed. This is helpful when designing generic functions that operate on arbitrarily sized arrays of any type. However, the previous examples would have problems with character, byte, or file type data. When defining parameters, information about the parameters' type and dimensionality can optionally be included in the parameters declaration list. The following example shows some of the ways to specify this information.

```
function min( x : numeric )

function min( x : double )

function min( x[*]:double)

function min( x[10][10]:double)
```

The first example, uses the keyword *numeric* to specify the parameter to be of any numeric type. In the second example, the parameter *x* can still have any dimension but it must be a double or coercible to a double. As table 3 showed, any numeric type is coercible to a double, so this effectively forces the function min to operate on numeric types only. The third example further constrains the input parameter by forcing it to be a one-dimensional variable.

To the right of the parameter name, a list of subscript-like items can be appended to the parameter name. These items express the desired dimensionality of the parameter. When an

asterisk appears between square brackets, the dimension can be any size. When a number is provided, the parameter is constrained to have a given size, as in the fourth example where the parameter *x* must be a two-dimensional variable with dimensions of size 10.

When this information is provided in the definition of a function, NCL verifies that any parameters passed to the function are of the appropriate type, dimensionality, and size. Therefore, if a user-defined function requires arrays of strings of a certain size, NCL can check the parameters passed to the function. When none of this information is provided, no checking is done.

There are two types of functions and procedures: those that are written in NCL, and those that are written in C or FORTRAN. From the user's point of view, they function the same. However, there are some things that functions and procedures written in C and FORTRAN can do that those written in NCL can't. To demonstrate this, consider the function *min* that returns the minimum of an array. An NCL function definition for *min* would look something like the following example.

```
function min ( x )
begin
        min = 9e99
        do i=0,totalsize(x)
                if x[i] LT min then min = x[i] endif
        endo
end
```

The difference comes when the parameter *x* is more than one dimension. NCL has no way to handle indexing of an array with an arbitrary number of dimensions. This doesn't mean that NCL functions won't accept parameters with an arbitrary number of dimensions, it just means that can not index these arrays automatically. The following NCL function will convert Fahrenheit to Celsius on arbitrarily sized arrays.

```
function f2c ( x )
begin
```

```
        f2c = (x - 32) * 5 / 9
end
```

Functions written in C or FORTRAN are defined differently. The details of an external function definition are covered in Section 5.5, "User extension to function set."

5.2.6 NCL flow control

NCL is equipped with loops and conditional statements for flow control. There are two kinds of loops: do loops and while loops. The syntax for a do loop is:

```
do i = 0,99,2 x = x + i endo

do i = 0,99
        x = x + 1
        y = x / i
endo
```

The first example is a one-statement loop that loops on the variable *i* at steps of 2 and executes the statement $x = x + i$. The *endo* keyword is necessary to instruct NCL that the last statement in the loop has been reached. The second example shows how to create a list of statements to execute at every loop iteration. This is essentially an extension of the first case.

The syntax for a while loop is:

```
while( i LT 99)
        x = x +1
        y = x/i
        i = i+1
endwhile
```

The *if* statement in NCL behaves just like *if* statements in other languages. The syntax is:

```
if (( i LT 99) and (y[i] GT i ))then
        x = x + 1
        y[i] = x/i
else
        x = x -1
endif
```

The keyword *if* is followed by a Boolean expression followed by the keyword *then* and a statement or statement list. At the end of the statement either an *else* or an *endif* keyword is needed. As in other languages, "lazy" evaluation of conditional expressions is used. In the above example this means that as soon as the first term of the conditional expression evaluates to false, the second conditional term is skipped because the entire expression evaluates to false. In the above example this means an potential error condition can be avoided by ordering the conditional terms from left to right.

5.3 Visualization Specification Block

The visualization specification block is the part of NCL the controls the graphics. It lets the user define a style of output plot and assign a name to it. When the user wants to display data in the plot style, the user calls the built-in procedures *assigndata* with the data and *display* with the name of the visualization specification block.

The contents of the visualization block are parameters to the graphics device. These parameters are the same as the resources for the HLUs defined in Section 4 of this document. For this example, actual HLU resources are not used; These are just examples.

```
visblk ncar mycontour {
        NgbPlotStyle :  CONTOUR
        NmpMap:         True
        NmpOutline :    PS
        NmpProjection : OR
        NvpX   :        .1
        NvpY   :        .1
        NvpWidth:       .8
        NvpHeight :     .8
}
```

The name of the block defined above is *mycontour.* The keyword *ncar* sets the graphics library to use. At first release only NCAR Graphics and *PolyPaint* will be supported. In the

future, other graphics libraries may be added. The syntax of the statements within the block is completely different than the rest of NCL. The syntax depends on the library chosen. For NCAR Graphics, the syntax is: resource name first, followed by a colon, followed by the value to set the resource to. For *PolyPaint*, the syntax will be based on the application command interface (aci) already implemented for *PolyPaint*. NCL does not even look he region between the braces; it is simply encoded into strings and passed to the graphics library for parsing. It is therefore possible for different graphics libraries to implement their parameter setting differently.

It is sometimes necessary to add or change the parameters in a visualization block. A special built-in procedure called *set* is used to accomplish this. For example, to set the data ranges and transformation, the following script could be used.

```
ncl> set ( mycontour,"NtiTitle", "Dataset 1")
ncl> set ( mycontour, "NcnXDataMin", -180);
ncl> set ( mycontour, "NcnXDataMax", 180)
ncl> set ( mycontour, "NcnYDataMin", -90)
ncl> set ( mycontour, "NcnYDataMax", 90)
ncl> set ( mycontour, "NcnXDimName", "lon" )
ncl> set ( mycontour, "NcnYDimName", "lat" )
ncl> assigndata(mycontour,file1.var1{level|1000}{frtime|0} );
ncl> display(mycontour)
ncl> frame
```

The next example shows how a sequence of frames could be created using NCL. In this example the same data are always plotted, but a new center of projection is provide for each iteration. This gives the effect of rotation around the globe. The display procedure does not require a data parameter because the data have already been specified in the assigndata call.

```
vsblk mycontour {
        . . .
}
set(mycontour,"NmpCenterLat", 0)
set(mycontour,"NmpCenterLon",0)
ncl> assigndata(mycontour,file1.var1{level|1000}{frtime|0} );
display(mycontour)
```

```
frame
do i = 1, 7
        set(mycontour,"NiMapCenterLon", ( i * 45))
        display(mycontour)
        frame
endo
```

This example shows how a visualization block can be changed and updated using NCL. These kinds of features should help in the production of animations.

5.4 Builtin functions and procedures

This section list some examples of built-in functions that will be available when NCL is released. Built-in functions are functions that are compiled and linked to NCL. Because of this built-in functions have no parsing overhead that functions written in NCL source have and have no data flow overhead for external functions. Furthermore, built-in functions have access to the NCL symbol table and can therefore accomplish a wider variety of tasks. This section represents an incomplete list.

5.4.1 GENERAL FUNCTIONS AND PROCEDURES

5.4.1.1 List - ls

The **ls** command provides the user with a listing of a directory. The user must provide the pathname to the desired directory.

```
ncl> ls("~ethan/datafiles")
aH.cdf
nT.cdf
...
```

5.4.1.2 Change directory - cd

The **cd** command changes NCL's current working directory of NCL.

5.4.1.3 Read formatted ASCII file - readfascii

The **readfascii** function reads formatted ascii into an NCL variable. This is used when the ascii file has a regular format, meaning each number has a constant number of characters, each column is separated by a constant number of white space characters, and rows are separated by a one-character delimiter. The user must specify the pathname of the file, the number of dimensions, the size of each dimension, and a format string.

```
ncl>var1 = readfascii("/u1/ethan/d.cdf",2,[2,4],"F4")
```

The ascii file for the preceding example could either:

```
5.000
1.000
3.000
10.00
1.000
22.00
2.000
30.00
```

or:

```
5.000 1.000
3.000 10.00
1.000 22.00
2.000 30.00
```

The resulting NCL array will be indexed as follows regardless of the type of file organization. NCL will read each number in order until it reaches an end of line. Based on the dimension number and size information, NCL automatically determines the appropriate index for each number. The following is a printout of the variable defined above.

```
ncl> print(var1)
(0,0) 5.0
(0,1) 1.0
(0,2) 3.0
(0,3) 10.0
(1,0) 1.0
(1,1) 22.0
```

```
(1,2) 2.0
(1,3) 30.0
```

5.4.1.4 Read binary fortran file - readbfortran

The **readbfortran** function reads a variable from a FORTRAN binary file. The user must specify the pathname, the number of dimensions, and the size of the dimensions. When **readfortran** is used, the order of the dimension sizes are in column-by-row ordering.

```
ncl> var1 = readbfortran("/u1/ethan/f.bin",4,[100,100,5,5])
```

5.4.1.5 Read binary C file - readbc

Analogous to **readbfortran**, except it reads from a C binary file. The dimension sizes are in row-by-column ordering

5.4.1.6 Read cray binary file - readcrayb

Reads and converts a file stored in Cray binary format.

5.4.1.7 Add NetCDF File - addnetfile

Since netCDF files easily map into the NCL data model, the **addnetfile** procedure will directly add a file record to the currently defined file record list.

5.4.1.8 Remove file - rmfile

Removes a file from the file record list.

5.4.1.9 Inquire about current variables - inquire

Prints out the contents of the current file record and variable list. Information about every variable in every file record is displayed.

5.4.1.10 Inquire about a specific variable - varinquire

Prints out the metadata information pertaining to a specific variable. These data include dimensions, sizes, storage types, and other associated attributes.

5.4.1.11 Write files - write, writeascii, writefort

Writes a variable to a file. **write** writes to a netCDF file. **writeascii** creates an ascii file for the data and a descriptor file for the variable's meta-data. **writefort** creates the same descriptor file, but the data file is in fortran binary.

5.4.1.12 Write a file record - writefile, writeasciifile, writefortfile

Writes a file record to a file.

5.4.1.13 Variable total size - size

Returns the total size of a variable or expression.

5.4.1.14 Dimension size - dimsize

Returns the dimension size for a dimension.

5.4.1.15 Variable minimum and maximum - min and max

Returns the minimum or maximum of an entire variable.

5.4.1.16 Remove variable - rmvar

Undefines a variable and frees memory used by that variable.

5.4.1.17 Render display - display

Built-in entry point to the graphics libraries. **display** takes a visualization specification block name and draws it onto the output frame.

C-2.

5.4.1.18 Assign data to visualization block - assigndata

Assigns data to a visualization block. These data are the data to plot. Resources in the visualization specification block set which dimension names in the data variable are used to select the spatial dimensions of the variable.

5.4.1.19 Create a plot overlay - overlay

overlay coordinates the plotting of two or more plots. The plots must be members of the same graphics library. The procedure takes two visualization block names. The first is called the base and the second is called the overlay. When one plot is an overlay, it inherits the viewport and data transformations of the base. When the base is displayed using the display procedure, the overlay plots will be drawn in the order in which the overlay procedure was called.

5.4.1.20 clear

Takes a visualization block name and clears the area of the output frame used by that plot.

5.4.1.21 quit

Exits NCL. If any variable has been changed or created, NCL will warn the user with a prompt that asks if they really want to quit.

5.4.1.22 loadscript

Loads and executes a script from a file. The file can contain function definitions, visualization blocks, and data manipulation commands.

5.4.1.23 Load a defaults file - load_defs

Loads a user defaults file. This file is described in Section 4 of this document, the High Level Utilities. The user defaults file

stores default resources for creating plots. To use the defaults file the user must call **load_defs** and provide a pathname to the file. Then any subsequent visualization specification blocks will use the defaults specified in the currently loaded file. The currently loaded defaults file can be changed without affecting the current visualization blocks. Once a visualization block is created, all of the applicable defaults are loaded and stored with the visualization specification block named.

5.4.1.24 Set prompt - setprompt

Changes the default prompt string from "ncl>" to some user specified string.

5.4.1.25 Read from keyboard - getstring

Retrieves a line of text from the keyboard.

5.4.1.26 Read a number from keyboard - getnum

Retrieves a number from the keyboard.

5.4.2 Built-in math functions

A wide range of mathematical data manipulation functions will be supported on first release.

5.4.2.1 Trigonometric functions

All trigonometric functions on a standard scientific calculator should be implemented.

5.4.2.2 Curve fitting

A least squares curve fit will be provided

5.4.2.3 Grid interpolation

Functions for interpolation from an irregular grid to a regular grid should be available, as well as interpolation from a regular grid of one size to the size and resolution of another.

5.4.2.4 Statistics

To be added.

5.5 User extension to function set

Users must be allowed to extend the function set. Some likely extensions are customized data ingestion methods and customized data manipulation functions. Users should be able to write their own functions in C or Fortran and be able to follow step-by-step instructions to add their function to the command language set. Addition of these functions should not require the user to have their own copy of the NCL executable in their home directory. This would unnecessarily waste user disk space.

Most likely, a support application called *nclextend* will be built for allowing the user to add to the function set. This application will prompt the user for a list of source files, include files, and libraries. An NCL function or procedure declaration also needs to be entered. Finally, the parameters from the NCL declaration need to be mapped into the C or FORTRAN function parameters. Since the NCL data model is different than the data model used in C and FORTRAN, the user must provide information about what part of the function's parameters will be passed to the C or FORTRAN function. For example, each variable in NCL holds all of the following information:

name	Name of variable
datatype	Datatype of variable
value	Actual value of variable
ndims	Number of dimensions

dimsizes[]	Array of dimension sizes
totalsize	Total number of elements in variable.
dimnames[]	Array of dimension names
dimvals[]	Array of coordinate value arrays
dimvardt[]	Array of the data types of the coordinate variables
natts	Number of attributtes associated with the variable
attnames[]	Array of attribute names
attvals[]	Array of attribute values
attdt[]	Array of attribute data

The following example demonstrates what adding a C function to the NCL function set might involve. Consider the function *min* written in C:

```
float min(var,size)
        float var[];
        int size;
{
        int i;
        float value = 9.9e99;

        for (i=0; i<size; i++)
                if(var[i] < value) value = var[i];
        return(value);
}
```

To add this function to the command set, the user must run the special application to add it. First the user is prompted for the NCL function declaration that will correspond to this function. It may look something like this:

```
function min(x:numeric)
```

The next step is to provide the pathname of the source file, any include files and any libraries needed.

The user must next enter the function or procedure name, the number of arguments, and the types of each argument. The user must then choose which parts of the input NCL parameters map to each parameter in the source function. For the current example, the only two parts of the input parameter *x* that need to be mapped to the source interface are the *value* and the *totalsize* fields. The user must specify that the *value*

part of the parameter *x* maps to the source parameter *var*, and the *totalsize* part of the *x* parameter maps to the *size* parameter in the source function. A GUI will be provided so little to no memorization is needed to provide this information.

Once this information has been entered, the *nclextend* application will compile the source into a module callable by NCL. This module will communicate with the NCL command interpreter either through sockets or through remote procedure calls. This is not the most efficient way of passing data but it allows the user code and the NCL application code to be separate entities and portable at the same time.

Users who want to maintain the same efficiency that the built-in functions have, can compile their functions directly into the NCL source. This, however, requires either the system administrator to reinstall the NCL software or the user to maintain their own copy of the NCL executable in their home directory. This procedure will not be outlined until a more thorough design has been completed. Having two options for extending NCL will provide systems and site administrators the flexibility to make their own policies in this matter.

SECTION 6

User interface requirements and design

NCAR Interactive's proposed Graphical User Interface (GUI) is a plot tool, an interactive program that allows users to make frames of single or multiple plots. This tool should also allow users to explore their data graphically. The GUI should be a WYSIWYG tool for producing publication-quality graphics.

This section of the functional specification has five main parts. The first is a complete description of the requirements for the user interface: what the user interface must be able to do. The second part is an initial design specification of the user interface: how we intend to provide the functionality specified in the requirements section. This document specifies the GUI design because the user interface defines the functionality of the entire application. The third part addresses style considerations that must be taken into account for a GUI. The fourth part describes how to use defaults files. The fifth part describes the use of Color Palettes.

6.1 Functional specification of NCAR Interactive's GUI

The functional specification of NCAR Interactive's GUI can be organized into some general requirements. First, the GUI must be usable by a wide variety of users with diverse graphical requirements. Second, it must be standardized so users of other similar applications will find it intuitive. Third, it must allow the user to simply create plots and specify information for those plots including the data specification for that plot. Lastly, it must allow the user to explore data in an intuitive manner.

6.1.1 Usability of NCAR Interactive's GUI

A user interface communicates information from the program to the user and communicates information from the user to the program. If the user interface does not perform these functions in a clear and concise manner, the application will be useless. Therefore, the primary objective of NCAR Interactive's GUI is to communicate its information clearly and in a straightforward manner. Likewise, user interaction with the program should be equally straightforward; it does no good to display the information clearly if the user can't make modifications in a clear way.

Complicating this task is the fact that NCAR Interactive must be able to service a wide variety of tasks for users with different skill levels. NCAR Interactive's user interface should allow a novice user to create a simple plot. Likewise, it should allow a more sophisticated user to create extremely complex plots. The user interface should be a useful tool for both levels of users. To accomplish this goal, the interface should generally have two levels. Initially, it should only request the bare minimum information necessary to create a plot; however, it should have a second level that displays every single piece of configurable information. This model allows the novice user to use the system without becoming overburdened with excess information. Also, it is more reasonable to require a user who wants to do more sophisticated plots to learn more about the application.

6.1.2 Adhering to standards

The primary requirements of the GUI for NCAR Interactive is that it be intuitive and easy to use. To accomplish this, the GUI should strictly comply with the Motif Style Guide since it will be a Motif application. Also, additional standards for text entry and accelerator keys need to be defined. This is extremely important since this GUI will most likely be only the first of a number of GUIs written to support NCAR Graphics. The

decisions made in this area should be used in later GUIs as well, so users of this application will not need to learn conflicting methods to use this package. It is critical that good long-range decisions be made at this early stage of the interface design.

6.1.3 Plot and data specification

As a plot builder/viewer, the GUI must make it easy for users to specify the information needed to create plots. This includes using scrollbars, dialogs, and menus to input information. Users find it much easier to choose their desired values from options, than to have to remember the value and then type it in. This also includes deciding on reasonable defaults for these values so the beginning user can simply let things default. Also, to support the wide range of NCAR Graphics users, it should be possible for users to customize default values for each of these fields.

Data specification has nearly the same requirements as plot specification. In fact, the ability for the users to specify their own set of defaults is even more important in this context because of the incredibly wide variety of data formats. Since it would be next to impossible for NCAR Interactive's GUI to anticipate all the different data requirements, NCAR Interactive will allow users a limited ability to configure this. However, it is still vitally important for NCAR Interactive to come up with data defaults that make sense for most users so novice users can still use the program.

6.1.4 Data exploration

Data exploration is one of the other main goals of this application. By using the interactive capabilities of an X interface, it should be possible for the user to specify regions of data directly, by simply pointing at a region of interest on a plot. The application should make this type of interaction simple and easy to use. The user should be able to use

techniques such as these to view data points from the plot or set data points in the plot.

6.2 Initial design specification for NCAR Interactive's GUI

It is often useful to separate the different functions of an application into separate windows of a GUI. For NCAR Interactive, we will initially organize the interface with the following five windows:

- main window
- plot specification window
- data specification and manipulation window
- region selection and data viewing window
- help window

Consider the descriptions of these windows as descriptions of the GUI's functionality; the windows will probably not look like this when NCAR Interactive is released.

The main window controls the application, allows the user to manipulate the layout of plots, and is eventually used to display the plots in a single frame. The plot specification window sets the characteristics of each plot (such as line width, color, etc.). The data specification and manipulation window allows the user to specify variables to display in each plot. In addition to spatial coordinates, it is possible to specify a sequence of data (often called timesteps) for each plot. It also allows the user to modify and create variables. The region selection and data viewing window is used when the user selects points or areas on the main window when the frame is being displayed. It allows the user to interactively view the data within the selected region and to apply that region selection to other variables. (This will be explained further later.) The help window provides descriptions of the application to aid the user's understanding.

6.2.1 Main window

Figure 3 shows that the main window contains a menu bar, a work area, and a small control panel area.

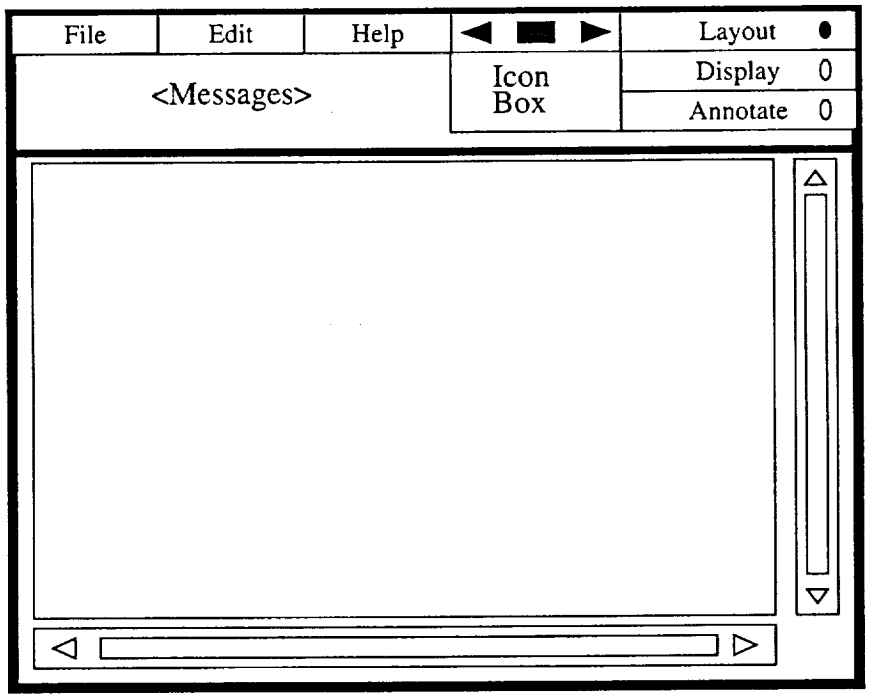


FIGURE 3 main window

Each of these components support the three basic functions of the main window. The main window is used to interactively set the layout of the output frame, to display the frame, and to allow annotation of the frame.

To do this, the main window will have three distinct user states. The first state is the plot layout mode. In this state, the user should be able to directly manipulate plots within the frame. The second state is the plot display mode. In this state, the main window will allow the user to view the plots on the screen and interactively explore the data being displayed in each plot. The third state is the frame annotation mode. In this

mode, the user can draw anything in the window using point-and-click drawing tools. Additionally, it will be possible for the user to have multiple occurrences of the main window.

6.2.1.1 Menu bar

The menu bar will have three submenus: the File menu, the Edit menu, and the Help menu. The File menu contains buttons that perform actions that apply to the entire application like saving the current state. The Edit menu is usually only used during plot layout mode; it contains the buttons necessary to bring up the plot specification window and the data specification and manipulation window. The Help menu is used to bring up the help window in one of its many different states.

6.2.1.1.1 File menu

The File menu contains the following buttons that perform the stated functions:

1. New	Completely clears all data and plot specifications currently in application, allowing the user to start over.
2. Open...	Reads in a previously saved session.
3. Load Defaults...	Loads in a global plot defaults file.
4. Save	Saves the current session.
5. Save As...	Saves the current session in a specific filename.
6. Print...	Prints the current plot and allows the user to specify print parameters before actually printing.
7. Exit	Exits the application.

6.2.1.1.2 Edit menu

All of the options in this menu automatically put the main window into plot layout mode; each option then performs the stated function:

1. New Plot	Creates a square bounding box in the workspace that represents the extent of a new plot. It also brings up the plot specification window with the new plot selected.

2. Change Plot	Brings up the plot specification window with the selected plot. This button is inactive if there is no plot currently selected.
3. Delete Plot	Deletes the currently selected plot. Before it does this, it brings up a dialog box to inform the user that this is a destructive action to insure that the user doesn't destroy a plot accidentally.
4. Copy Plot	Copies the currently selected plot. First it puts up another bounding box to represent the new plot and then brings up the plot specification window with the new plot.
5. Undo	Reverses the last action of the user. Actions that are not reversible must first warn the user of that fact.
6. Overlay...	Brings up a dialog box that allows the user to select a base plot for the new overlay from the current plots. It also allows the user to select one of the current plots for an overlay of the base plot or allow the user to create a new plot to be used as the overlay plot.
7. Edit Data	Brings up the data specification and manipulation window .

6.2.1.1.3 Help menu

The Help menu contains buttons to access the help facility:

1. On Context	Turns the cursor into a question mark, allowing users to click on the item they don't understand. The help window comes up and displays the appropriate help screen.
2. On Window	Displays the help window with the help screen that is appropriate for the main window in question.
3. On Help	Displays the help window with the help screen that describes the help interface in question.
4. On Keys	Displays the help window with the help screen that describes the accelerator keys and mouse actions in question.
5. On Version	Displays the help window with the help screen that contains version information on the application in question.

6.2.1.2 Work area

The work area of the main window has two basic functions. It is used to manipulate the plot bounding boxes when the main window is in plot layout mode, and it is used to display the

plots when the main window is in plot display mode. It will be a virtual area. In other words, users can use as much space as they want in the work area, if they want to use more area than is currently visible due to the size of the window, scroll bars will be provided.

6.2.1.3 Control panel area

The control panel area is in the upper right corner of the main window. It has two different controls. The first is a toggle box that toggles the main window between plot layout mode, plot display mode, and frame annotation mode. The second control is a frame advance control. It will allow the user to move forward and backward through all of the plots as a group.

6.2.1.4 Plot layout mode

In this state, the user can move and resize plots. Each plot within the output frame will be represented by a bounding box. The bounding box should textually display minimal information about the plot. Users will be able to directly change the size and placement of each bounding box, therby changing the size and placement of the actual plot when it is visualized.

The user will be able to directly manipulate attributes of each plot by selecting the plot with M1 (mouse button 1) and then selecting the Change Plot button from the Edit menu, which would bring up the plot specification window for the selected plot. There will also be a keyboard accelerator associated with the Change Plot menu option; this will allow users to bring up the plot specification window for a selected plot without having to use the mouse. Additionally, it will be possible for the user to simply double-click with M1 on the plot they wish to edit to bring up the plot specification window. This redundancy is useful to make the application as user-friendly as possible.

The user will be able to specify and modify the data associated with each plot by using the data specification and manipulation window. This window will be brought up by a variety of methods similar to those used in the plot specification window. First, the user can select any single plot with M1 and then select the Edit Data button from the Edit menu to bring up the data specification and manipulation window for the selected plot. There will also be a keyboard accelerator associated with the Edit Data menu option. Also, it will be possible for the user to simply double-click with M2 (mouse button 2) on the plot for which they wish to modify the data. Again, all this redundancy makes the application easier to use.

6.2.1.5 Plot display mode

In this state, the plot or plots within the output frame will be displayed. It also will allow the user to move forward and backward through timesteps of the plots. In addition, the user will be able to change the timestep for specific plots, leaving the rest of the plots on the same timestep.

It will also be possible to interactively explore the data while in this mode by selecting data regions with the mouse. This will be accomplished by users selecting the region they are interested in, or by simply clicking on the single data point they are interested in. The main window will work in conjunction with the region selection and data viewing window to show the values of the variables in the selected region and allow additional region selection tools.

6.2.1.6 Frame annotation mode

In this state, the plots within the output frame are displayed as a reference point for the annotation. It allows the user to add any text or supported graphics to the output. This annotation will show within the window until deleted; it does not depend upon the particular frame currently being viewed. It can be

thought of as a colored transparency placed over everything else.

6.2.2 Plot specification window

The plot specification window (Figure 4) will be used to set the characteristics of the specified plot. It will be displayed

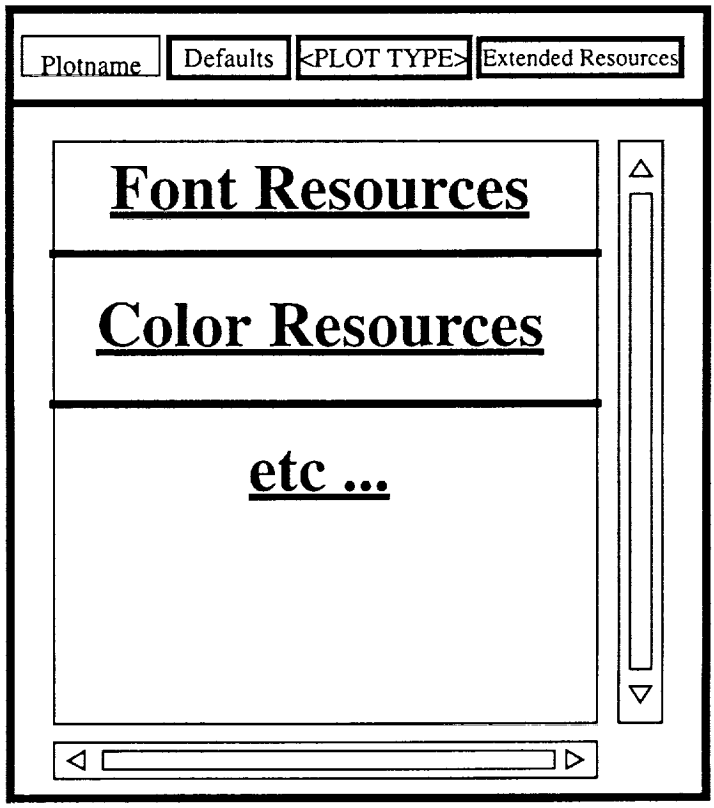


FIGURE 4 Plot specification window

when the user specifies a plot to edit and then uses one of the actions described earlier to bring up the plot specification window.

The plot specification window is a dynamic window. Each occurrence of it may look slightly different. This is to accommodate the many varieties of plots the user can create.

The user will be able to change the name and set the type of plot from within the plot specification window, but the exact characteristics that are configurable from within the plot specification window will depend upon the plot type the user selects, since different plot types have different configuration options. Additionally, a Data Input button will be available at all times. This button will bring up the data specification and manipulation window for the currently selected plot.

There will also be a toggle button available for the user to select *extended resources*. This will allow the user to toggle between the original state of the window and the extended resources state of the window. When the window first comes up, it will allow configuration of basic plot resources that are most often changed by users. When the user selects the extended resources state of the window, it allows the user to configure all plot resources. Because there are so many plot resources, and because many of them are difficult to understand and used only infrequently, this model will hide the more obscure resources from the user until they really need or want to manipulate them. It is hoped that this will keep the application easy to use for beginners while still allowing more experienced users the ability to do more complex things.

The Defaults button will be described in "Default files" on page 114.

6.2.3 Data specification and manipulation window

The data specification and manipulation window (Figure 5) will be used to set the data inputs for each plot. It will be used

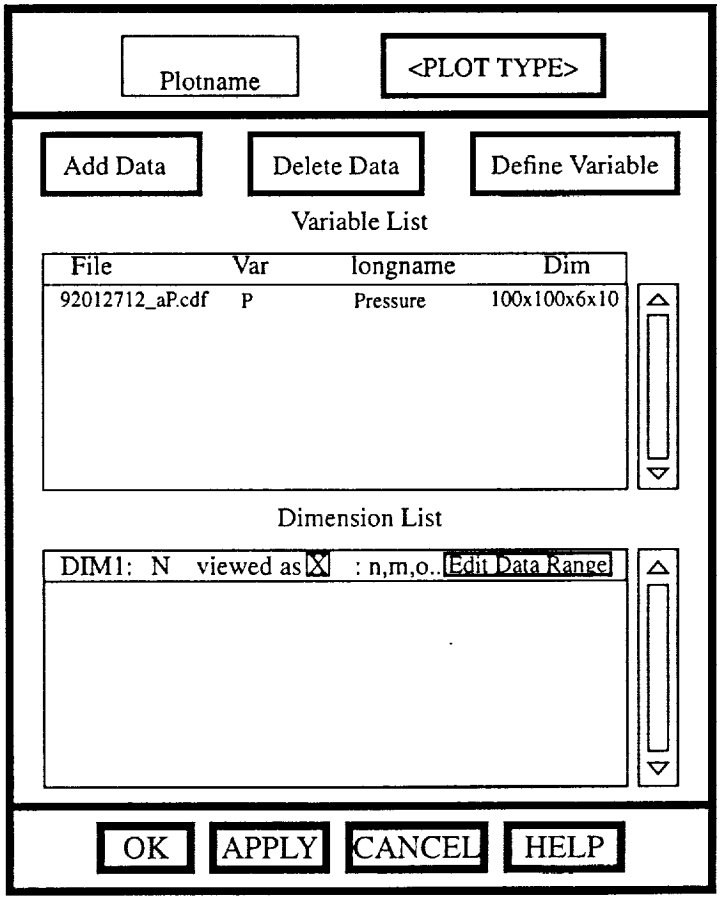


FIGURE 5 Data specification and manipulation window

to specify the data to be used within the selected plot. It will be displayed when the user specifies a plot and then uses one of the actions described in the main window section.

The data specification and manipulation window has two basic functions. First, it allows the user to manipulate data by reading data in from a file or by actually creating data. Second, it associates this data with a given plot. Since both of these functions will take place in the same window, the data

specification and manipulation window can come up in one of two states. If there is no currently selected plot, the data specification and manipulation window will come up partially inactive. The part of the window used for importing data will be available, but the portion of the window that is used for associating that data with a plot will be inactive.

6.2.3.1 Defining data

To allow the user to read data into the GUI, the user will use the Add Data button in the data specification and manipulation window. This will bring up a data selection box (Figure 6) which looks similar to the Motif file selection box. This

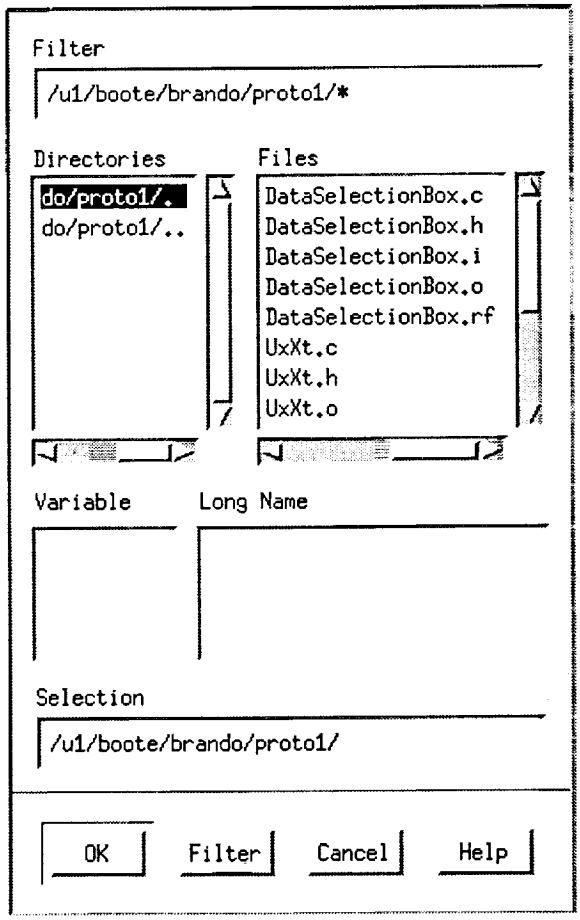


FIGURE 6 data selection box

window will enable the user to select files in the same way as the Motif file selection box, but it will also display the variables contained within each datafile (as long as the file is in a known format). From here the user can simply select variables from the window to import into the GUI. After the variables are imported, they can be associated with a plot.

The user can also create his own variables. This is done by pressing the Define Variable button. This will bring up a window with an interactive NCL session. Each variable that is currently defined within the GUI will be available to the NCL session, and any variable instantiated while in the NCL session can be imported to the GUI.

6.2.3.2 Associating data with a plot

Once data has been imported into the GUI, it is available to be used within a plot. To do this, the user must have a plot selected. The name of the selected plot will be displayed in the upper left corner of the data specification and manipulation window. Then the user can select the variables they want displayed in the selected plot from the list of current variables. Once this is done, the dimensions of the variable will be shown in the scrolled window below the variables. The GUI will either determine the dimension names from the data, or the user will be prompted to provide dimension names. Then the user can bind each dimension of the data to a spacial dimension of the plot, or the user can specify any given dimension to remain static. Once the user has successfully bound the X and Y axis (and Z for 3-D plots) the data can be plotted.

The user can also use only particular ranges of the variables' dimensions. To do this, they would select the Edit Data Range button on the line of the dimension they would like to modify.

This will bring up the data range window (Figure 7), which

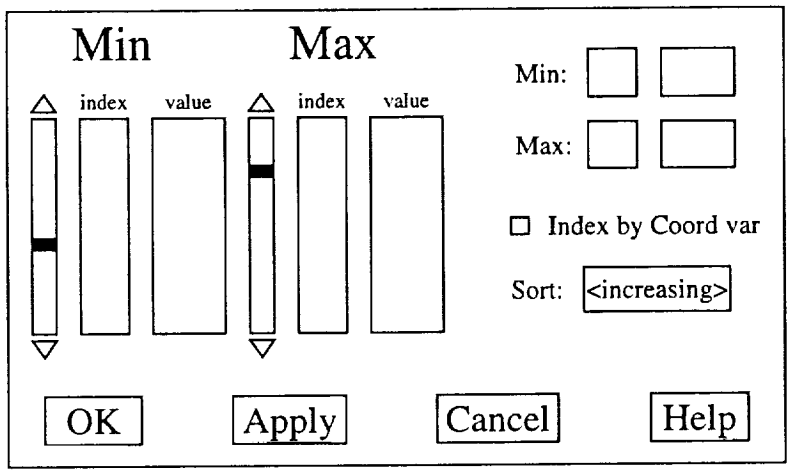


FIGURE 7 data range window

allows the user to sort each dimension and select any contiguous range of the dimension to be used in the plot. By default, the plot uses the entire range of each dimension.

The data range window shows the minimum and maximum value that exists within the dimension in the upper right corner. In the lower right corner, the user can use the Option menu to determine if and how the data is sorted. On the left side of this window, the user can use the scrolled lists to select the data range to be used in the current dimension by selecting the maximum and minimum values to be used.

6.2.4 Region selection and data viewing window

The region selection and data viewing window (Figure 8) will be used in conjunction with the main window during plot

display mode. It will be used to view the data values that are

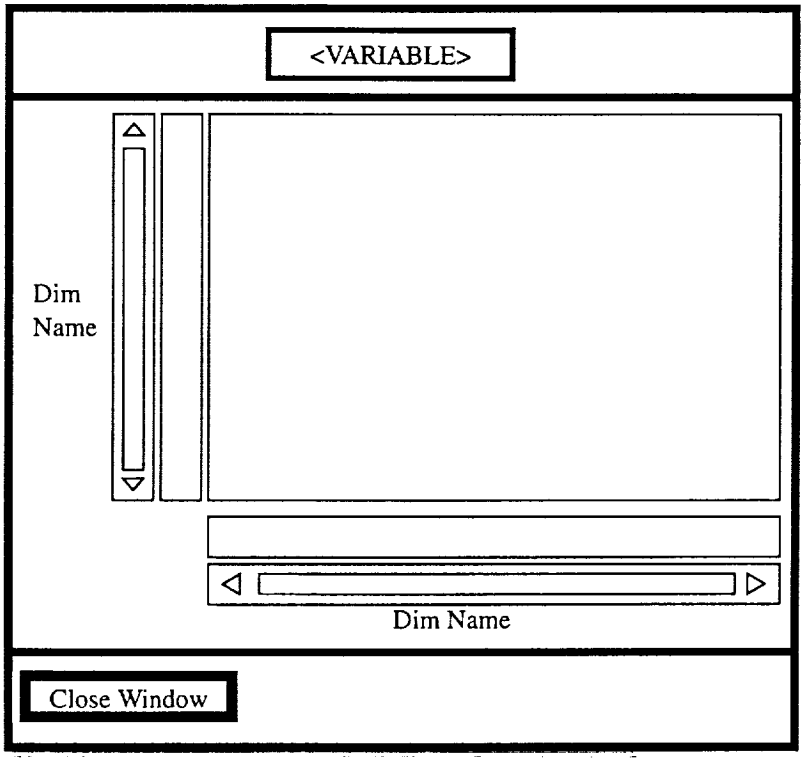


FIGURE 8 Region selection and data viewing window

currently being displayed in each plot. It will allow the user to rubber-band a region within the main window and allow the user to see the actual data values at the selected regions. There will be a Format button in this window that will bring up a window that will allow the user to specify the format in which to display the variable values.

Additionally, if the user double-clicks on a value, the user should be allowed to edit that value by either replacing it with the defined missing value in the data, or by any value they enter. The user will be allowed to have multiple instances of the region selection and data viewing window. Each instance will correspond to a different bounded region of the frame. This will allow the user to compare different regions of data.

6.2.5 Help window

The help window (Figure 9) allows users to recieve help on any aspect of the application. The top portion of the window

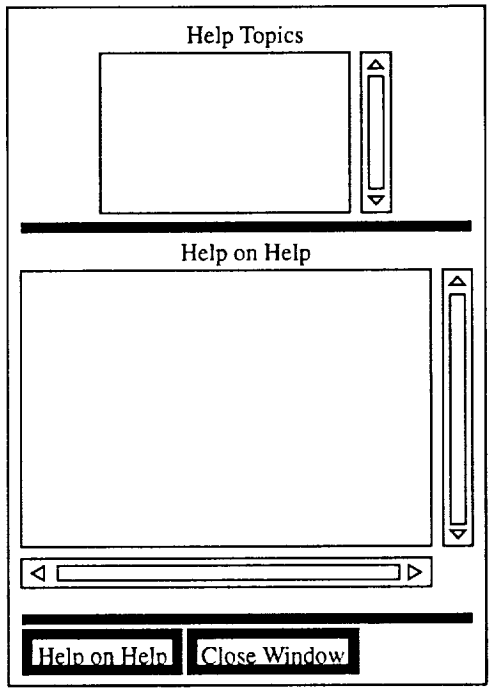


FIGURE 9 help window

will contain a scrolled list of help topics available on the application. The bottom portion of the window will contain a scrolled text window with the text for the currently selected help topic.

With this type of configuration, users will be able to browse through the help screens simply by clicking on topics that interest them. Likewise, if they are having problems with a particular portion of the application, they can press the On Context button in the Help menu of the main window. This will change the cursor to a "?" and allow them to click on the item they are having problems with. Once this has been done, the help topic associated with that portion of the application

will automatically be selected in the help window. The Help on Help button within the help window automatically selects the help topic associated with the help window.

6.3 Style considerations

This section of the document will be used to indicate our choices for an NCAR Graphics GUI style guide. This will be in addition to the Motif Style guide for GUIs written in Motif. These choices may need to be modified in the future to support other toolkits such as the Open Look widget set. At this time, we are adopting all the behavior restrictions described in the Motif Style guide as well as the following button bindings, key bindings, key accelerators, and mouse actions.

6.3.1 Button bindings

To be added after showing prototypes to users.

6.3.2 Key bindings

To be added after showing prototypes to users.

6.3.3 Key accelerators

To be added after showing prototypes to users.

6.3.4 Mouse actions

To be added after showing prototypes to users.

6.4 Default files

Default files are extremely important to the use of NCAR Interactive. By using the GUI, the user will be able to create and load default files that can also be used with NCL and the HLUs.

The user will be able to load a general defaults file that will apply to all plots he creates from that time forward. This is called the global defaults file. The user will also be able to load a defaults file directly into the specification of a single plot. This is known as a plot-specific defaults file. With this approach, the user can find a defaults file that works best for most of his plot resources (global), and then use the specific defaults file to change particular resources for a specific plot.

6.4.1 Global defaults

The main window is the part of the GUI that will deal with the global defaults files. From the main window, the user will be able to load a global defaults file by using the Load Defaults button in the File menu. This will only be necessary if the user doesn't like the "system defaults" that will come with NCAR Interactive. The global defaults will only apply to the current main window. In other words, if the user wants multiple main windows, each one will have its own global defaults.

6.4.2 Specific defaults

The plot specification window is the part of the GUI that will deal with the specific defaults files. From the plot specification window, the user will be able to load a defaults file into the current plot specification window. This will update all the resources in the current window to reflect the new defaults. Additionally, the user can take the current configuration of the window and write it out as a defaults file.

6.5 Color palettes

To be filled in after prototyping.

SECTION 7 Detailed Resource Descriptions

This section contains useful detailed descriptions and diagrams of the resources listed in Section 4 (High Level Utilities). This section has not been updated with the most current information on contour, vector, and streamline resources. This section will not continually keep pace with the evolution of NCAR Interactive code, but it will be updated when the code is frozen for each release.

7.1 Tick Marks, Tick Mark Labels and Grids

These resources are shared by all 2D utilities and can be used to control tick mark configuration.

7.1.1 NtmXGroup & NtmYGroup

These resources allow the user to combine both top and bottom or left and right tick mark configurations so that only one set of resources for each axis is needed. When these resources are set, only the left and bottom tick mark resources are used to configure the tick marks.

7.1.2 NtmMajorLineThickness & NtmMinorLineThickness

This sets the line thickness for the major and minor ticks. Since tick marks belong to a given viewport, the thickness is set as a fraction of the maximum viewport range.

7.1.3 NtmMajorLength & NtmMinorLength

Sets the length of the tick marks. This length is expressed as a fraction of the maximum viewport range.

7.1.4 NtmMajorLineColor & NtmMinorLineColor

Sets the tick mark line color to a specific color index.

7.1.5 NtmMinorPerMajor

Sets the number of minor tick marks per major tick mark.

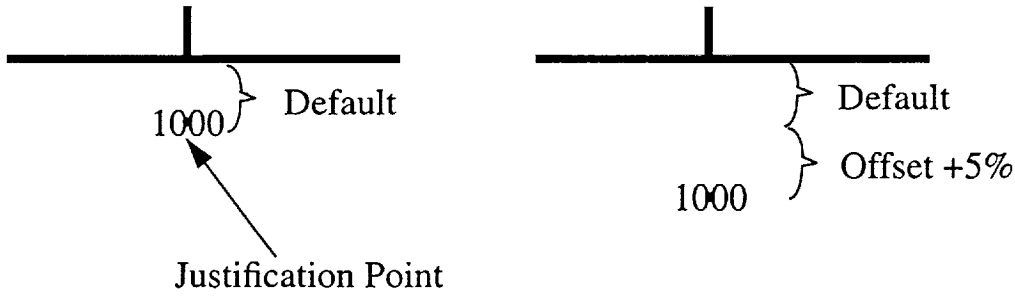
7.1.6 NtmLabelFont

Specifies what font to use for tickmark labeling. Related resources are:

NtmLabelFontSize
NtmLabelFontAspect
NtmLabelFontColor
NtmLabelTextJust
NtmFontThickness
NtmLabelTextAngle

7.1.7 NtmLabelOffset

This is the distance as a fraction of the maximum viewport range, where all of the label justification points for the axis are placed.



7.1.8 NtmBorder

Specifies how the plot or graph is outlined. The outline settings should be:

none
perimeter (4 sides)
Two-sided (left and bottom axes)
Two-sided (Top and bottom axes)
Two-sided (left and right axes)
Two-sided (left and top axes)
Two-sided (right and top axes)
Two-sided (right and bottom axes)
Three-sided (no bottom axis)
Three-sided (no left axis)
Three-sided (no top axis)
Three-sided (no right axis)

The following resource configure the border:

NtmBorderLineDashPattern
NtmBorderLineColor
NtmBorderLineDashLength
NtmBorderLineThickness

7.1.9 NtmAxisControl

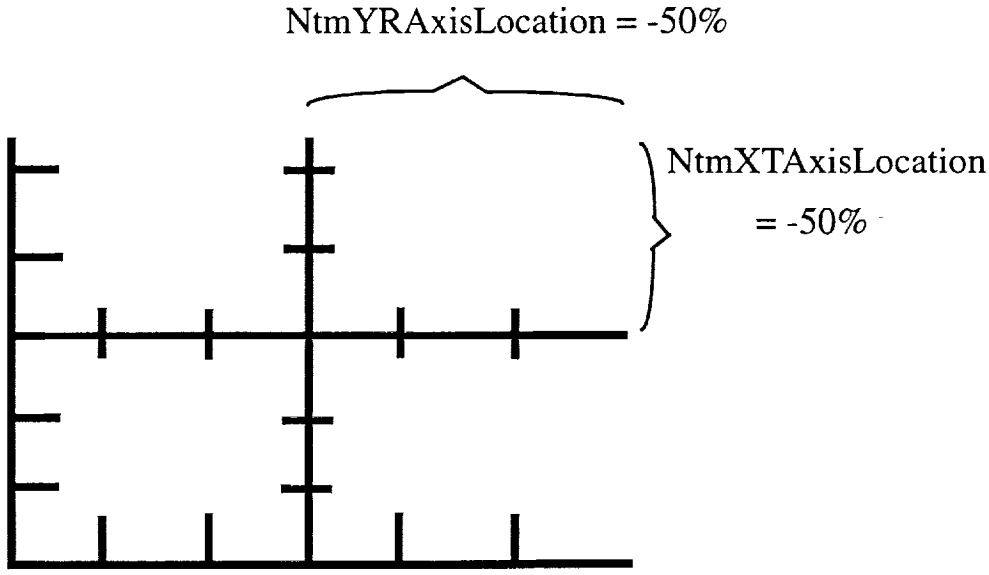
Specifies characteristics of an axis. The settings are

Do not draw line portion of axis, only draw tick marks
Do draw line portion of axis and tick marks

7.1.10 NtmAxisLocation

Specifies the location to move an axis as a fraction of the viewport range. This parameter is useful for moving the

bottom axis, for example, to the center of the viewport and plotting a curve that intersects the x axis for some value of y.



7.1.11 NtmMajorGrid

Switches on the grid option. A grid is drawn connecting major tick marks on opposite sides of the plot.

The following configure the major grid lines:

NtmMajorGridLineDashPattern
NtmMajorGridLineDashLength
NtmMajorGridLineColor
NtmMajorGridLineThickness

7.1.12 NtmMinorGrid

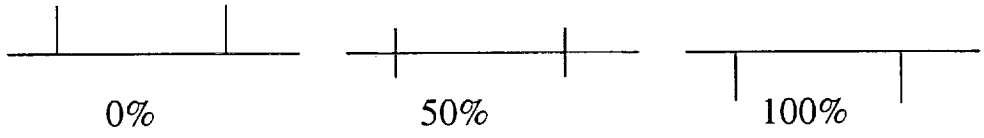
Switches on the grid option. A grid is drawn connecting minor tick marks on opposite sides of the plot.

The following configure the minor grid lines:

NtmMinorGridLineDashPattern
NtmMinorGridLineDashLength
NtmMinorGridLineColor
NtmMinorGridLineThickness

7.1.13 NtmMajorOutwardLength

Specifies the length of the outward portion of the major tick marks as a fraction of NtmMajorLength. The following shows this.



7.1.14 NtmMinorOutwardLength

Same as NtmMarkMajorOutwardLength.

7.1.15 NtmLabelFormat

Specifies the type of axis labeling for linear-style plots.

None
Use scientific notation [-] [i] [.] [f] x 10 e
Use exponential notation
Use no-exponent notation ("normal" numbers)

7.1.16 NtmLabelFormatExponent

This parameter is used to specify the value of the exponent and length of *i* above.

Any float value is the valid range
Other wise NCAR Graphics decides default value

7.1.17 NtmLabelFormatFraction

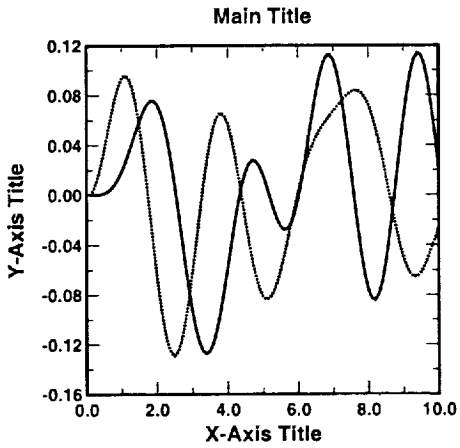
This parameter is used to specify the length of *f* above.

Any float value is the valid range
NCAR Graphics decides the default value

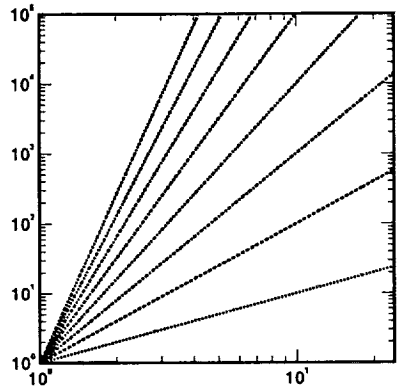
7.1.18 NtmStyle

Tick marks come in a variety of flavors; therefore a tick mark style resource is needed to distinguish between the types. Four basic styles are found in tick marks, log, linear, time and geographic tick marks. Log and linear tick marks are fairly straightforward. However, time and geographic tick marks are not. Time tick marks display units of time, everything from minutes to years. Geographic tick marks display units of degrees, minutes, and seconds as well as north, south, east, and

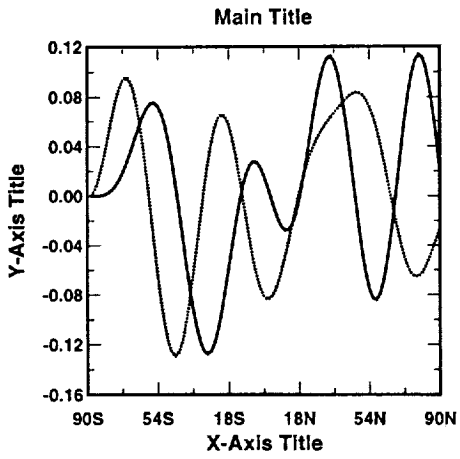
west. Below are examples of the various types of tick marks.



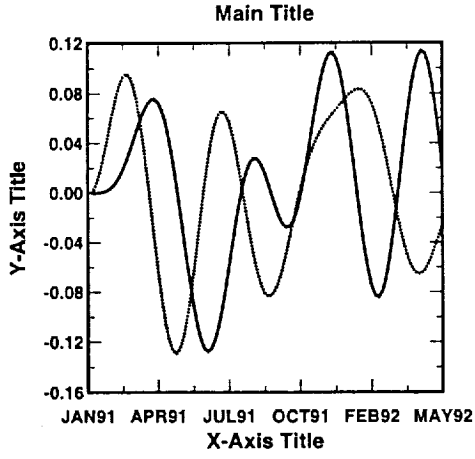
Linear tick marks



Log tick marks



Geographic tick marks



Time tick marks

Since time and geographic tick marks are special the resource names have either NtmGeo or NtmTime in front of the resource name. Log and Linear tick marks share the same resources. The following list of resources covers Log and Linear specfic resources first and then Time and Geographic. Each heading lists what mode the resource can be used in. When no mode is provided in the heading then the resource can be used in any mode.

7.1.19 NtmMode

There are four modes that tick marks can operate in. In automatic mode tick marks are picked automatically. In manual, the start end and spacing are specified by the user. In explicit, an array of values in data coordinates is provided by the user, and a tick mark is placed at every data point in the array. This provides a way of completely customizing the tick marks. The final mode is none, where no tickmark are drawn at all. A separate tick mark mode is needed for each axis in the plot.

7.1.20 NtmMaxTicMarks - Automatic Mode

This is used when the tick marks are in automatic mode. It represents the maximum allowable tick marks. Most likely the actual number of tick marks will be less.

7.1.21 NtmStart - Manual Mode

Used when in manual mode. Sets the starting point for the major tick marks. If this is less than the minimum data point for the viewport, the viewport data mapping is not changed and the first tick mark will appear at the first tick location where s*n > datamin. Where s is the spacing between major tick marks and n is an integer multiplier representing the number of the tickmark. If the start is greater than or equal to

the minimum data point in the viewport, then the tick marks begin at NtmStart.

7.1.22 NtmEnd - Manual Mode

Sets the ending point for the major tick marks. The conditions for where the ending tickmark is placed are the reverse of the starting point.

7.1.23 NtmSpacing - Manual Mode

Sets the interval between major tick marks.

7.1.24 NtmSpacingType - Manual Mode

Specifies what method is used to compute positions of the major tick marks. In the examples below *i* is the tickmark number, with zero being the first and the last one satisfies the following condition $MajorTick_i <= end$. Start, spacing and end are NtmStart, NtmEnd and NtmSpacing.

$MajorTick_i = start + spacing * i$
$MajorTick_i = start + spacing * 10^i$
$MajorTick_i = start + (spacing)^i$

7.1.25 NtmMinorStart - Manual Mode

Sets the beginning point for minor tick marks. The default is that the minors start at the same point as the major tick marks.

7.1.26 NtmMinorEnd - Manual Mode

Sets the ending point for the minor tick marks.

7.1.27 NtmValues - Explicit Mode

This is the resource for the explicit mode of tick mark operation. NtmValues is an array of data points where tick

marks are drawn. NtmValueIndex can be used to index individual elements of the NtmValue array.

7.1.28 NtmNumTicks - Explicit Mode

This is the number of array elements provided in the Ntm marks array.

7.1.29 NtmLabelsText - Explicit Mode

This is an array of strings used to label each of the tick mark values provided by the NtmValue resource. NCAR Graphics function code may be encoded into the string to provide configurability of the output string. If a Null string is present no label will appear for the corresponding tick mark. NtmLabelIndex indexes the NtmLabelsText array.

7.1.30 NtmMappingOrder

Specifies an increasing or decreasing mapping order on the axis.

7.1.31 NtmGeoTickStyle

When geographic labeling style is selected there are several styles for representing geographic coordinates. First, is degrees only. In this case every tickmark is labeled with a real number representing degrees and an N,S,W or E depending on what the NtmGeoAxis parameter is set to for the axis. Second the real number of degrees is converted to an integer number of degrees followed by the number of minutes, followed by an N,S,W or E. Finally, a degrees, minutes, and seconds display.

When geographic tickmark style is used, it is possible that the tick start is greater in value than the end. This happens when the start and end wrap around. This may be limited to the HLU being used

This tickmark style can only be used with non-mapped plots. See the mapping composite resource for labeling mapped plots.

7.1.32 NtmGeoTickAxis

For each axis whether the axis is latitude or longitude is specified.

7.1.33 NtmTimeTickStyle

When time labeling style is selected, several styles can be labelled. The first thing set is what the units are that separate major tick marks. These can be seconds, minutes, hours, days, weeks, months, and years.

The time tick style is very difficult to specify since various formats can be used to express time.

The simplest case is when the input data is regularly spaced time indices. For this case a base time for the first data element and a time spacing need to be specified.

The hardest case is when the input data is irregularly spaced and some user convention has been used to specify the time coordinates of the data.

7.1.34 NtmTimeDataBase

This is the time in year, month, day, hour, minute, and seconds. This value corresponds to the left corner of the viewport for the X-axis and the bottom corner for the Y-axis. This is not the first tickmark value. This option can be used when data is regularly spaced. Even for Automatic mode time tick generation NtmTimeDataBase and NtmTimeDataSpacing must be set.

7.1.35 NtmTimeDataSpacing

The amount of time between each data point.

7.1.36 NtmTimeTickStart - Manual

The time value in year, month, day, hour, minute and seconds of the first tickmark.

7.1.37 NtmTimeTickEnd - Manual Mode

Ending time value.

7.1.38 NtmTimeTickSpacing - Manual Mode

Spacing of tick marks in years, months, days, hours, minutes and seconds.

7.1.39 Explicit Mode Time Ticks

Explicit mode time tick marks rely heavily on the 2D data transformation used by the HLU. Further research on how time coordinates are used for irregular and regularly spaced data is needed. Explicit mode requires an array of data coordinates which represent locations to place tick marks. Data coordinates that make up 2D data transformations are usually real vaul numbers. Converting from some user convention to the data transformation does not appear to be trivial at this time.

7.2 LEGENDS

See legend description in the Annotation section.

7.3 LABELBARS

See labelbar description in the Annotation section.

7.4 TITLES

Titles are special kinds of text items in which the position of the text is defaulted to be at some fixed location, above, below, right or left of the plot. In the case of X and Y titles the location of the title will be guarenteed not to interfere with the tickmark labels. Resources for small adjustments up/down and left/right are provided as well as a resource for flipping the rotation of a title. This is particularly useful for the Y axis title. It is often a matter of taste how the Y axis title is positioned. The following subsections list only resources for Main titles. This is because the same type of resources are used for X and Y titles.

7.4.1 NtiMainText

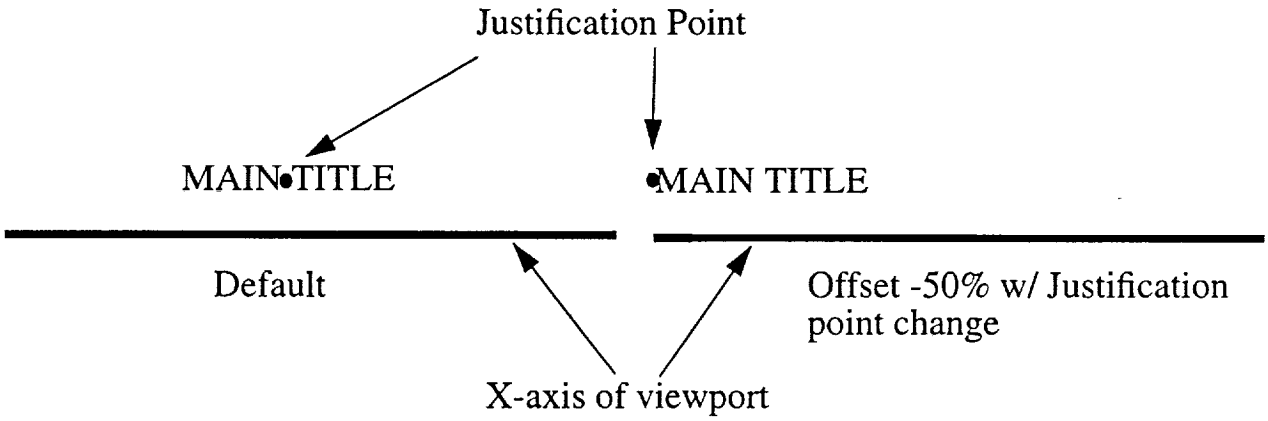
Sets text used to create main title. NCAR Graphics function codes can be used to configure output string. See NCAR Graphics 3.0 Manual.

The following resources set the remaining attributes for the main title:

NtiMainFont
NtiMainFontSize
NtiMainColor
NtiMainTextJust
NtiMainFontAspect
NtiMainFontThickness
NtiMainTextAngle

7.4.2 NtiMainXOffset

Moves the title from its default position by some offset in the X direction. This value is a percentage of the X axis of the viewport.



A negative offset is to left and a positive is to the rigtht.

7.4.3 NtiMainYOffset

Moves the title from its default position by some offset in the Y direction. This value is a percentage of the Y axis of the viewport.

7.4.4 NtiMainPosition

Can be set to top or bottom for main titles and X Axis. For Y titles it can be left or right.

7.5 MAPS

Maps can only be used in the 2D utilities CONTOUR, VECTOR and STREAMLINE. In these utilities maps can either be on or off.

The following resources define the layout and projection of a map.

7.5.1 Nmp

Turns on and off the map options.

7.5.2 NmpProjection

This attribute determines which type of projection is used to display a map. The value of NmpProjection can be one of ten types. See the NCAR Graphics User's Guide Version 2.00 (pp. 206-210) for a detailed description of these map types.

LC - Lambert conformal

ST - Stereographic

OR - Orthographic

LE - Lamber equal area

GN - Gnomonic

AE - Azimuthal equidistant

SV - Satellite-view

CE - Cylindrical equidistant

ME - Mercator

MO - Mollweide-type

The default is CE.

7.5.3 NmpTicks

This turns on and off the map tick mark option. When on tick marks are drawn around the perimeter of the map where grid lines intersect the perimeter.

Related resources are:

NmpTickLength
NmpTickThickness
NmpTickColor

7.5.4 NmpTickLabels

This turns on and off the labeling of the map tick marks.

Related resources are:

NmpTickLabelFont
NmpTickLabelFontColor
NmpTickLabelFontSize
NmpTickLabelFontAspect
NmpTickLabelFontThickness
NmpTickLabelFontTextAngle
NmpTickLabelTextJust

7.5.5 NmpTickLabelStyle

Select different ways of labeling geographical tick marks. There are three styles to choose from: Degrees only, degrees and minutes, and degrees, minutes and seconds.

7.5.6 NmpGrid

This attribute specifies whether or not to turn the grid on. The grid consists of latitude and longitude lines at spacing specified by NmpGridSpace.

The default is "on".

7.5.7 NmpGridDotSpace

This parameter specifies the distance between points used to draw a grid in degrees. The default is 1.0. The values must fall between .001 and 10.

7.5.8 NmpGridSpace

This parameter specifies the grid spacing in degrees. A zero will turn the grid off. The default is 10 degrees. See the NmpGridDashPattern attribute for the dash pattern definition of the grid lines.

7.5.9 NmpUserLinesDotDist

This attribute specifies the distance between dots along a dotted line drawn by MAPIT.

The default value is 12 (out of 4096).

7.5.10 NmpPlotterResltn

This attribute specifies the width of the target plotter in plotter units. The value is used for calculating distances, for example. The default is 4096.

7.5.11 NmpUserLinesDashPattern

This attribute specifies the dash pattern to use for lines drawn by the MAPIT utility. It is a 16-bit value. The default is 21845 (0101010101010101 binary).

7.5.12 NmpUserLinesType

This is a parameter which specifies whether or not lines drawn by the MAPIT utility will be dotted or solid. Dotted lines are affected by the NmpUserLinesDotDist attribute and solid lines are determined by the NmpUserLinesDashPattern attribute.

The value is either "dotted" or "solid". The default is "solid".

7.5.13 NmpPerimLineDashPattern

This sets the line attributes for the map perimeter.

Related resources are:

NmpPerimLineDashLength
NmpPerimLineColor
NmpPerimLineThickness

7.5.14 NmpGridLineDashPattern

This sets the line attributes for the latitude and longitude grid lines.

Related resources are:

NmpGridLineDashLength
NmpGridLineColor
NmpGridLineThickness

7.5.15 NmpLimbLineDashPattern

This sets the line attributes for the limb lines.

Related resources are:

NmpLimbLineDashLength
NmpLimbLineColor
NmpLimbLineThickness

7.5.16 NmpCOLineDashPattern

This sets the line attributes for the Continental Outlines.

Related resources are:

NmpCOLineDashLength
NmpCoLineColr
NmpCOLineThickness

7.5.17 NmpUSLineDashPattern

Sets the line attributes for the US state outlines.

Related resources are:

NmpUSLineDashLength
NmpUSLIneThickness
NmpUSLineColor

7.5.18 NmpPOLineDashPattern

Sets the line attributes for the political country outlines.

Related resouces are:

NmpPOLineDashLength
NmpPOLineThickness
NmpPOLineColor

7.5.19 NmpLimbLine

This is a boolean parameter which is either "on" or "off". "on" specifies that a limb line should be drawn on the map; "off" specifies that it should not. The default is "on".

7.5.20 NmpOutlineType

This parameter determines the types of outlines used in a map.

NO - No outlines are generated

CO - Continental outlines are generated

US - U.S. state outlines are used.

PS - Continental, international, and U.S. state outlines are used.

PO - Continental and international outlines are generated.

7.5.21 NmpOutlines

This parameter specifies whether or not outline dashpatterns are on or off.

7.5.22 NmpElliptical

This parameter allows a user to create a map inscribed within an ellipse. The bounds of the ellipse are set by the normal rectangular perimeter. This parameter can be either "on" or "off". The default is "off".

7.5.23 NmpLabels

This parameter determines whether or not to label the meridians, the poles, and the equator.

The value may be "on" or "off". "on" is the default.

Related resources are:

NmpLabelsFont
NmpLabelsFontSize
NmpLabelsFontColor
NmpLabelsFontThickness
NmpLabelsFontAspect

7.5.24 NmpMinVectorLength

Points closer than this value to a previous point are omitted. The default value is 4 (out of 4096).

7.5.25 NmpPerim

This attribute specifies whether or not to draw a perimeter around a map. The default is "on".

7.5.26 NmpSatDistance

This parameter determines how many Earth radii the satellite is from the center of the earth.

7.5.27 NmpSatSight2Center

When NmpSatDistance is > 1, this parameter measures the angle between the line to the center of the earth and the line of sight. If this value is 0, the projection shows the earth as seen by a satellite looking straight down (the "basic view").

The default value is 90.

7.5.28 NmpSatU2Projection

When NmpSatSight2Center is non-zero, this parameter measures the angle from the positive u axis of the basic view to the line OP, where O is the origin of the basic view, and P is the projection of the desired line of sight. This value is positive when measured counter clockwise.

7.5.29 NmpProjectionOrigin

This is an array of 2 elements which specify the coordinates of the origin of projection. The first element is the Latitude and the second is the longitude.and the second is the longitude.

The default value is 0, 0.

7.5.30 NmpProjectionRotation

This is the angle of rotation of a projection. The default value is 0.

7.5.31 NmpRectLimitType

This parameter specifies how to interpret the rectangular limits (NmpRectLimit1, NmpRectLimit2, NmpRectLimit3, NmpRectLimit4) of a map. The type can be one of the following: MAX, CORNERS, LIMITS, ANGLES, EXPLICIT.

MAX - The maximum useful area produced by the projection is plotted. The limit parameters are not used.

CORNERS - (Limit1, Limit2) and (Limit3, Limit4) represent opposite corners of the map. The Limits represent coordinates in latitude and longitude.

LIMITS - The limits 1-4 specify the min and max values of U and the min and max values of V respectively.

ANGLES - Limits 1-4 represent positive angles in degrees which define the angular distances from a point on the map to the left, right, bottom, and top edges of the map, respectively.

EXPLICIT - Limits 1-4 are two-element arrays giving the latitudes and longitudes in degrees of the four corners of a map.

For a detailed explanation of these parameters, see the discussion of the MAPSET parameters in the EZMAP section of the NCAR Graphics User's Guide Version 2.00.

7.5.32 NmpRectLimit1

see above

7.5.33 NmpRectLimit2

see above

7.5.34 NmpRectLimit3

see above

7.5.35 NmpRectLimit4

see above

7.5.36 NmpBoundMask NmpBoundMaskSize NmpBoundMaskType

These three parameters specify what map boundaries to mask. NmpboundMask is an array (size NmpBounMaskSize) of map area identifiers which are masked out if NmpBoundType is "masked", or it is an array of map area identifiers which are not masked (everything else is) if NmpBoundType is "unmasked".

In this manner, if the user wants to mask out the boundaries around Cuba, for example, NmpBoundMask would equal 602, NmpBoundMaskSize would equal 1, and NmpBoundMaskType would equal "masked".

If, on the other hand, the user wants to mask out everything except Cuba, NmpBoundMaskType would equal "unmasked".

7.5.37 NmpGridMask NmpGridMaskSize NmpGridMaskType

These three parameters specify what areas grid lines will be masked or unmasked over. These parameters work in the same fashion as the NmpBoundMask parameters, except grid lines are masked instead of political boundaries.

7.5.38 NmpLineMask NmpLineMaskSize NmpLineMaskType

These three parameters specify what areas lines drawn by the user will be masked or unmasked over. These parameters work

in the same fashion as the NmpBoundMask parameters, except user lines are masked instead of political boundaries.

7.5.39 NmpAreaFill NmpAreaFillSize NmpAreaFillType NmpAreaFillColor NmpAreaFillPattern

These fill parameters are similar to the above masking parameters except that the color of the fill areas and the pattern of the fill are supplied in the NmpAreaFillColor and the NmpAreaFillPattern arrays, respectively. The size of these arrays is NmpAreaFillSize.

NmpAreaFill is an array of map area identifiers which specify areas to be filled (if NmpAreaFillType is "fill") or areas not to be filled (if NmpAreaFillType is "nofill").

NmpAreaFillColor is an array of indices into a color table which specifies the color of each area. If no color array is provided, the high level utility will automatically generate one.

NmpAreaFillPattern is an array of fill flags. Each element in the array takes on one of the following values:

1 solid

2 parallel lines

3 criss-crossed orthogonal lines

4 lines at varying angles and spacings

5 dots? circles? other other shapes or patterns?

1 is the default.

7.5.40 NmpLatLabels, NmpLonLabels

These parameters detail how to label longitude and latitude lines on a map projection.

If, for example, your map projection is a complete globe with the center of projection at 0 degrees longitude and 0 degrees latitude, the latitude lines will intersect the perimeter at several locations around the globe. The longitude lines, on the other hand, will intersect the perimeter at only two points on the projection: the North and South Poles.

Labeling the latitude lines where they intersect the perimeter of the globe is not a problem, but labeling the longitude lines at the perimeter intersection is; all the labels will overlap at the poles.

Two labeling schemes are provided to address this problem: labeling on the perimeter where the grid lines intersect the perimeter and labeling on the map along a line of constant latitude or longitude. This latter method allows the user to pick the equator, for example, as a line of constant latitude along which the longitude labels will appear. In this manner, the user can label the latitude lines at the perimeter and the longitude lines along the equator or some other line of latitude.

This resource specifies which labeling method to use for the latitude and longitude labels, respectively. Either resource (NmpLatLabels, NmpLonLabels) can have any of these values.

"off" - turn off labeling

"perimeter" - label at the point where the grid line intersects the perimeter

"constant_grid" - label along a constant grid line

7.5.41 NmpLatLabelConstant, NmpLonLabelConstant

These parameters are used only when a NmpLatLabels or NmpLonLabels are set to the "constant_grid" type. If NmpLatLabels is "constant_grid", then NmpLatLabelConstant

should be set to the value of constant LONGITUDE along which the latitude labels will appear.

Likewise, if NmpLonLabels is set to "constant_grid", then NmpLonLabelConstant should be set to the value of constant LATITUDE along which the longitude labels will appear.

The NmpLatLabelConstant can be any valid real longitude value (-180.,+180.), and the NmpLonLabelConstant can be any valid real latitude value (-90., +90.).

0. is the default for both parameters.

7.5.42 NmpLatLabelStyle, NmpLonLabelStyle

These parameters specify the format for latitude and longitude labels. They may be in one of three formats.

"degrees" - The label is a real number in degrees

"minutes" - The real number of degrees is converted to an integer number of degrees followed by the number of minutes.

"seconds" - A degrees, minutes, seconds display.

The default is "degrees".

7.5.43 NmpLatLabelSpace, NmpLonLabelSpace

These parameters set the spacing in degrees between labels. The default is the NmpGridSpace value.

7.5.44 NmpLatLabelFont, NmpLatLabelFontSize, NmpLatLabelFontThickness, NmpLatLabelColor, NmpLatLabelFontAspect, NmpLonLabelFont, NmpLonLabelFontSize, NmpLonLabelFontThickness,

NmpLonLabelColor, NmpLonLabelFontAspect

These parameters set the characteristics of the text, such as size, color, font, etc. These attributes are analagous to the corresponding text resources. ges are 1.

7.5.45 Miscellaneous

A MAPIT function and Great Circle function will need to be provided for drawing lines and arcs between points.

7.6 XYPLOT

An XY plot is a two-dimensional plot of zero or more curves. The functionality is similar to that found in the NCAR Graphics Autograph utility.

An XY plot can have any number of curves. For each curve, a line style, color, pattern, etc. must be specified or defaulted. These attributes are passed as arrays of attributes. The size of the array is determined by the number of curves in the plot (NxyNumCurves).

7.6.1 XY Plot General Parameters

The following parameters represent the arrays of attributes for plot curves.

7.6.1.1 NxyNumCurves

This is an integer which specifies the number of curves which will be plotted.

1 is the default value.

7.6.2 XY Plot Curve Parameters Arrays

The following parameters are arrays of size NxyNumCurves. if using the array scheme for defining curves. Each curve in a plot has one element from each array which specifies a particular curve attribute. For example, if you were plotting 3 curves, curve 1 would be defined by the attributes in the first element of each of the following arrays, curve 2 and 3 would be defined by the second and third elements, respectively.

If using the index scheme for defining curves, the following parmeters effect a single curve which is specified by the NxyLineIndex parameter.

7.6.2.1 NxyLineStyle

This attribute specifies the types of curves to use in an XY plot. The three possible types are Auto, Dashed, and Labeled.

If Auto (the default) is chosen, then XYPLOT HLU will automatically generate curves which are solid lines interrupted periodically by a letter of the alphabet.

If Dashed is chosen, then the user must pass a dash pattern, which is defined below. The XYPLOT HLU will use this dash pattern to represent the plot curve.

If Labeled is chosen, then the user must pass a string which represents the line label. This attribute is also defined below. The XYPLOT HLU will use the string pattern to represent the plot curve.

7.6.2.2 NxyLineIndex

This attribute specifies the index of the current curve in an array of curve data. When setting other resources, such as line thickness or color, these resources will apply to the line/curve specified by the NxyLineIndex. When using this indexed scheme to change the attributes of other curves, the

NxyLineIndex must be changed to the corresponding curve number before setting the attributes for another curve.

The default value is 1, the first element of an array of curve data.

Related resources are:

NxyLineThickness
NxyLineColor
NxyLineSmooth
NxyLineDashPattern
NxyLineDashLength

7.6.2.3 NxyLineLabelText

This is an array that holds the line labels for each curve. If nothing is set and line labeling is turned on then the curve are labeled by their order in the list of curves.

NxyLineLabelFont
NxyLineLabelFontSize
NxyLineLabelFontThickness
NxyLineLabelAspect
NxyLineLabelFontColor

7.6.3 XY Plot Control Parameters

These parameters control the display of the plots, such as how to interpret the data with implied coordinates, and how to interpret missing values. Some of these parameters will be arrays of values when using the array scheme for defining curves, and some will be scalar values when using the indexed scheme for defining curves.

7.6.3.1 NxyScheme

This parameter determines how the attribute resources are interpretted - either as arrays or single values.

If NxyScheme is "array", then the attribute resources, such as NxyLineLabelText and NxyLineLabelFontColor, are treated as arrays of size NxyNumCurves. Each element of the array

will apply to a separate curve. i.e. the first element applies to the first curve and so on.

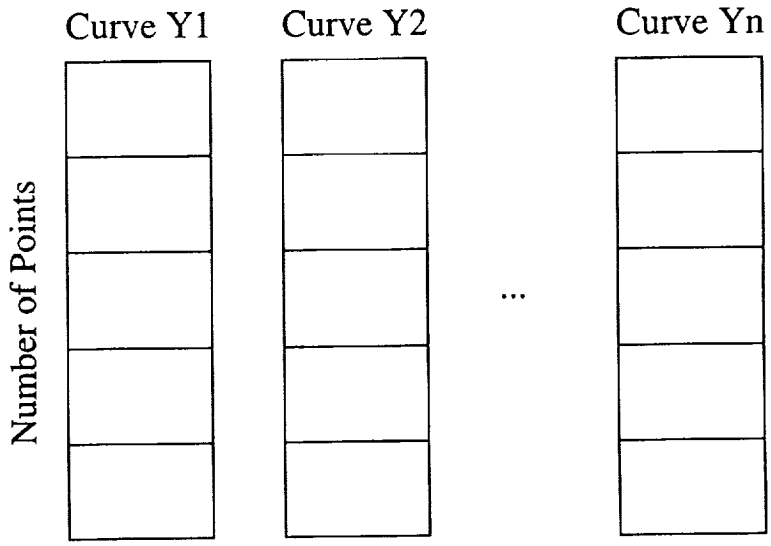
If NxyScheme is "index", then the attribute resources are treated as single values. The value of NxyLineIndex determines which curve or line the attribute resources apply to.

7.6.3.2 NxyControlPlotType

This parameter specifies the format of the input data.

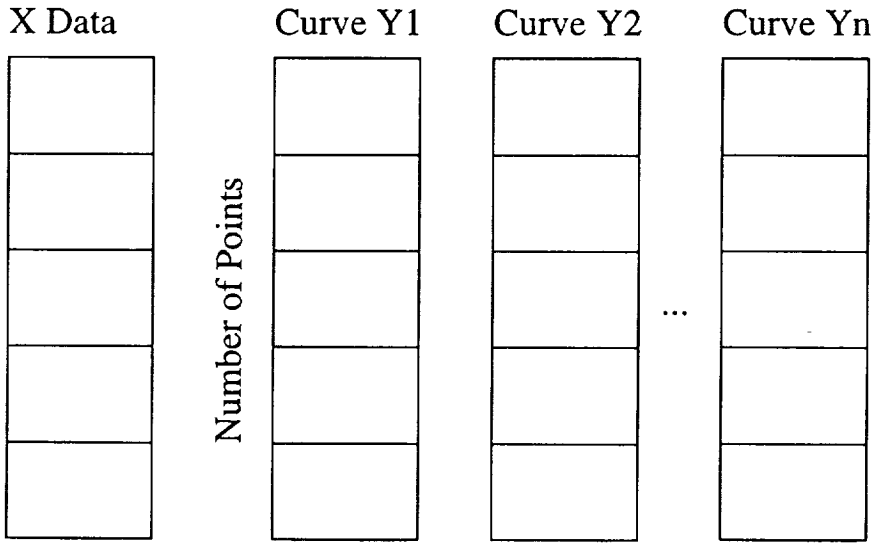
The data can be in one of six formats.

1. The data is one or more singly dimensioned arrays (one per curve) of Y data values. The X coordinates are implied; they go from 1 to the number of points defining a curve. Each of the Y arrays has a size of NxyLineNpoints.

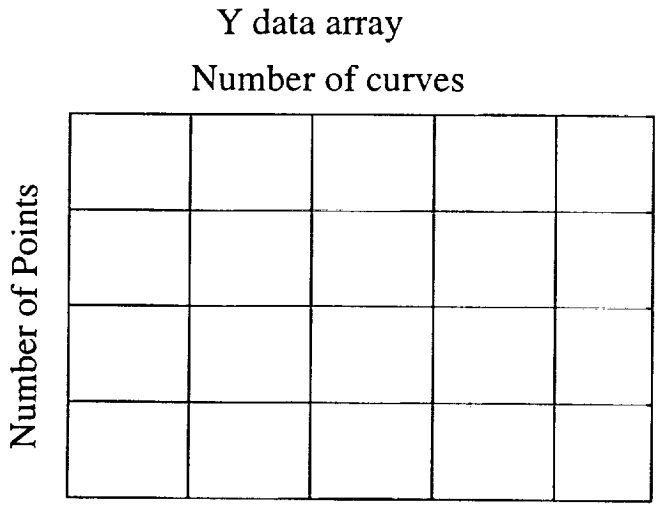


2. The data is one or more singly dimensioned arrays (one per curve) of Y data values, as above, and one singly dimensioned

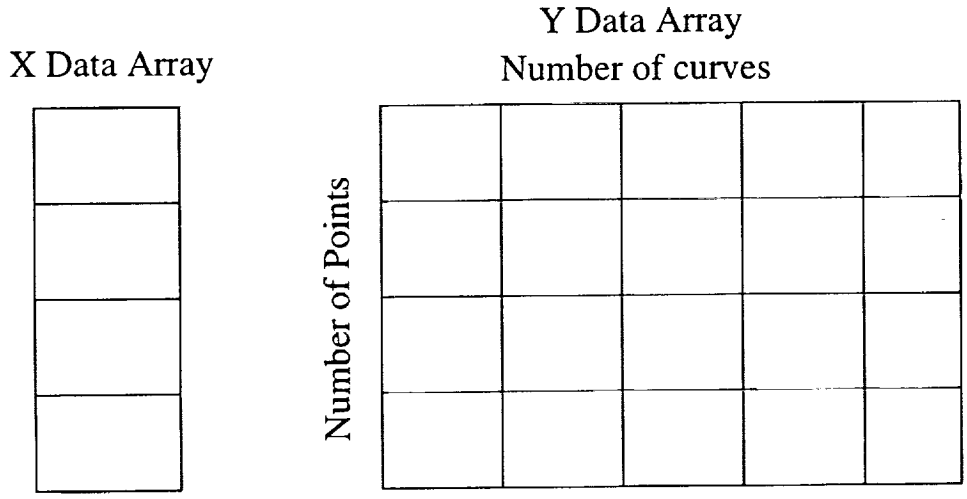
array of X data values. In this plot type, all of the curves share the same X data points but different Y data values.



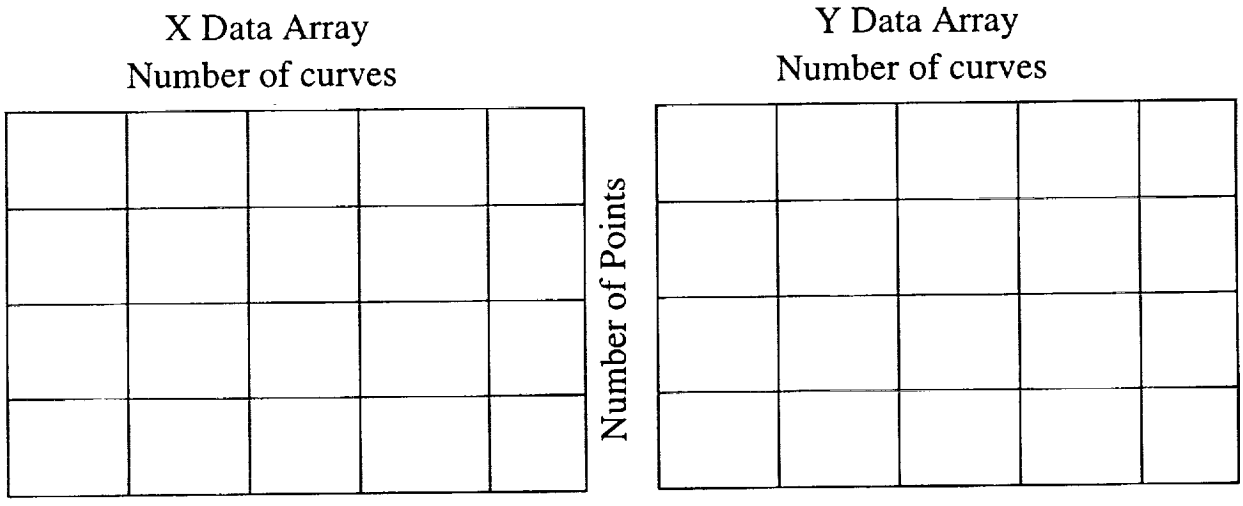
3. The data is one doubly dimensioned array of Y data values. One dimension of the array is size NxyLineNpoints, and the other is NxyNumCurves. As in case 1 above, the X data is implied.



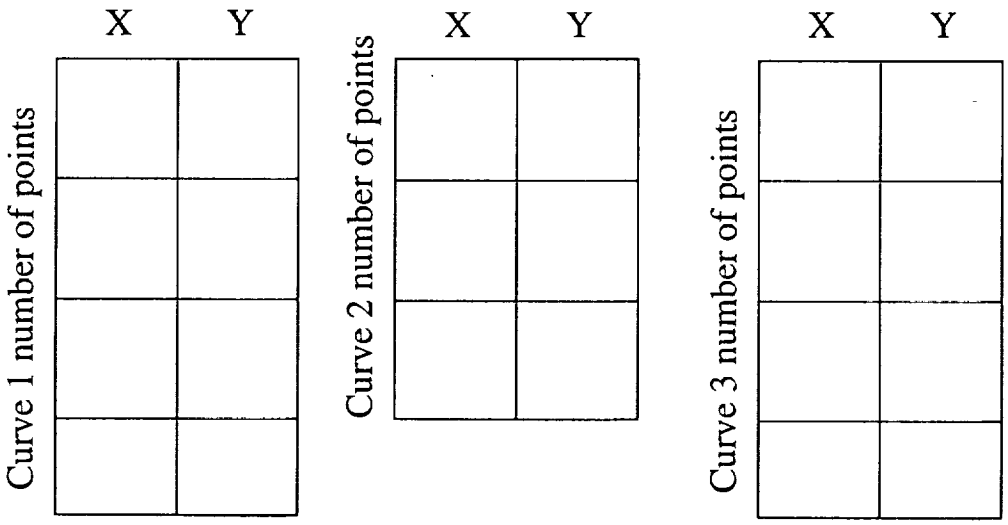
4. The data is one double dimensioned array of Y data values, as above, and one singly dimensioned array of X data values.



5. This data format is 2 doubly dimensioned arrays of X and Y data. Both arrays are dimensioned NxyLineNpoints by NxyCurveNum.



6. This data format consists of arrays of XY pairs. In this case there is one curve per array, but each array can have a different number of points. Each array is 2 dimensional, where one dimension is the number of points and one dimension is 2 (X and Y).



7.6.3.3 NxyYLine1, NxyXLine1, NxyYline2, NxyXLine2, ...NxyYlineN, NxyXlineN

These parameters are the X and Y data arrays. Their dimensions and ordering depend upon the value of NxyControlPlotType, NxyControlRows, and NxyControlOrder.

Depending on the value of NxyControlPlotType, N may range from 1 to NxyNumCurves.

For example, if NxyNumCurves = 3, and NxyControlPlotType = 2, then there will be four data arrays defined:

NxyXLine1 - the X data points shared between all three curves

NxyYLine1, NxyYLine2, NxyYLine3 - the three Y arrays for each curve.

7.6.3.4 NxyControlRow

This specifies the dimensioning of curve data that has two dimensions (except for case 6 above).

1 the first dimension is curve number and the second is the point number.

2 the first dimension is the point number and the second is the curve number.

1 is the default.

7.6.3.5 NxyControlOrder

This parameter specifies what order the two-dimensional data is stored in, Fortran or C.

C - the last dimension varies fastest.

Fortran - the first dimension varies fastest.

The default is C. ??????

7.6.3.6 NxyControlInvert

This attribute allows the user to graph x as a function of y

0 normal graph

1 lets x = f(y) and thus invert the graph

0 default value

7.6.3.7 NxyControlWindow

This attribute allows user to omit curve portions falling outside the grid window.

0 No omission

1 Omit portions outside the grid window

0 default value

7.6.3.8 NxyControlMissingValue

This parameter defines missing values in data.

1.E36 is the default, but it can be assigned any value.

7.6.3.9 NxyControlXMin

Specifies the minimum X user coordinate.

NxyControlMissingValue - Autograph calculates the minimum for each graph

float value - specifies the minimum value to use.

NxyControlMissingValue is the default value

7.6.3.10 NxyControlXMax

Specifies the maximum X user coordinate.

Valid values and default are analogous to NxyControlXMin

7.6.3.11 NxyControlXSmallest

If NxyControlXMin is NxyControlMissingValue and therefore the XYPLOT HLU computes the minimum value, values less than NxyControlXSmallest will not be considered in the computation.

any float value is valid

NxyControlMissingValue is the default value (XYPLOT HLU computes it).

7.6.3.12 NxyControlXLargest

Analogous to NxyControlXSmallest.

7.6.3.13 NxyControlYMin NxyControlYMax NxyControlYSmallest NxyControlYLargest

The Y parameters are all analogous to the X parameters above.

7.7 CONTOUR

The contouring HLU should esentially implement the functionality of CONPACK without requiring the user to understand a lot of the messy details involved with coordinating the EZMAP, GRIDAL, AREAS, PLOTCHAR and LABELBAR utilities. This is a draft list of the resources needed to specify a contour plot. The CONTOUR HLU uses all of the common composite resources listed in section 4.3. The resources listed here are specific to creating , filling and drawing contours.

7.7.1 NcnDataType

Specifies whether the input array is integer, float or double.

7.7.2 NcnXDataGridType & NcnYDataGridType

The contouring utility will handle regular, non-cartesian regularly spaced, irregularly spaced and randomly spaced data. Regular grids can be defined as grids where each coordinate pair can be calculated by evaluating the following functions.

$$XCoordinate = X_0 + i * (X_{nx} - X_0) / nx$$
$$YCoordinate = Y0 + j * (Y_{ny} - Y_0) / ny$$

Non-cartesion regularly space grids are grids where there are not regular linear steps but the steps represent some linear function. An example of this kind of grid is a vertical profile of the atmosphere where the Y direction varies as a log scale of pressure. Coordinate values for non-cartesian regular grids can be computed by the following types of equations.

XCoordinate = FX(i)
YCoordinate = FY(j)

For random and irregular spaced grids are grids where there are no functional relationships between the coordinates that can be used to calculate the coordinates. The difference between random and irregular is that all points for a given Y index share the same X coordinate value and all points for a given X index share the same Y coordinate value.

7.7.3 NcnXDataDim

Specifies which dimension of the data array should be mapped to the X dimension of the output frame.

7.7.4 NcnYDataDim

Specifies which dimension of the data array should be mapped to the Y dimension of the output frame.

7.7.5 NcnXDataCord & NcnYDataCord

These are not used differently based on the type of grid. The resource is not used for regular grids. Instead the NcnXDataMax, NcnXDataMin, NcnYDataMax, and NcnYDataMin. For any other type of grid these hold all of the coordinate values for each index or in the case of random grids all of the coordinates for each point.

7.7.6 NcnXDataMax & NcnXDataMin

For regular grid, these set the beginning and ending X coordinates.

7.7.7 NcnYDataMax & NcnYDataMin

For regular grids, these set the the beginning and ending of the Y data coordinates.

7.7.8 NcnXStart

Used to specify the minimum coordinate that will be visible. This is the data point that maps to the left side of the viewport.

7.7.9 NcnXEnd

Used to specify the maximum coordinate that will be visible. This is the data point that maps to the right side of the viewport.

7.7.10 NcnYStart

Used to specify the minimum coordiate that will be visible in the Y direction. This is the bottom of the viewport.

7.7.11 NcnYEnd

Used to specify the minimum coordiate that will be visible in the Y direction. This is the bottom of the viewport.

7.7.12 NcnIntervals

Array of contour inervals. If this is unset intervals will automatically be picked.

7.7.13 NcnCIntervalFillPattern

Fill pattern for each interval.

7.7.14 NcnCIntervalColor

Fill color indices for each contour interval.

7.7.15 NcnCLine

Array of line attributes for each contour line. If an intervals line attributes are not set then not contour line will be drawn.

NcnCLineColor
NcnCLineDashPatern
NcnCLineThickness
NcnCLineSmooth
NcnCLineDashLength
NcnCLineLabelStyle

7.7.16 NcnCLineLabels

Array of labels used to annotate each contour line. If label resources are not set then no labels are drawn for that contour line.

NcnCLineLabelText
NcnCLineLabelFont
NcnCLineLabelFontSize
NcnCLineLabelFontThickness
NcnCLineLabelFontAspect
NcnCLineLabelFontColor

7.7.17 NcnNumCLines

Total number of contour lines.

7.7.18 NcnInfoLabelText

Informative label and associated attributes.

NcnInfoLabelFont
NcnInfoLabelFontColor
NcnInfoLabelFontSize
NcnInfoLabelFontThickness
NcnInfoLabelFontAspect
NcnInfoLabelTextJust
NcnInfoLabelTextPosX
NcnInfoLabelTextPosY

7.7.19 NcnHighLow

Turns on and of High/Low labels.

7.7.20 NcnHighText

Sets a string to be printed at every high point.

NcnHighText
NcnHighFont
NcnHighFontColor
NcnHighFontSize
NcnHighFontThickness
NcnHighFontAspect

7.7.21 NcnLowText

Sets a string to be printed at every low point.

NcnLowFont
NcnLowFontColor
NcnLowFontSize
NcnLowFontThickness
NcnLowFontAspect

7.7.22 NcnMissingValue

No contours will be drawn

7.7.23 NcnHLBox

Turns high low label boxs off or on.

NcnHLBoxWidth
NcnHLBoxHeight

NcnHLBoxLineThickness
NcnHLBoxLineColor
NcnHLBoxLineDashPattern

7.8 VECTOR

To Be Added.

7.9 STREAMLINE

To Be Added.

7.10 Common 3D Resources

These resources are common between the iso-surface and the surface utilities. Setting these resources precludes the setting of the corresponding surface and iso-surface resources unless the user wants them to differ between the two utilities.

7.10.1 N3dEyePoint

This attribute specifies the x,y,z coordinate position of the viewer's eye. This value should be outside the data space.

7.10.2 N3dViewCenter

This attribute specifies the coordinate position looked at. It should be inside the data space.

7.10.3 N3dContLines

This sets the contant lines to be used for drawing a surface. Valid values are:

U - draw lines of constant u

V - draw lines of constant v

W - draw lines of constant w

U and V - draw lines of constant u and v

U and W - draw lines of constant u and w

V and W - draw lines of constant v and w

U and V and W - draw lines of constant u and v and w

7.10.4 N3dSkirt

This attribute specifies whether or not to draw a skirt or wall along the edges of a surface where it intersects the data boundary.

It can be either "on" or "off"

7.10.5 N3dContourHigh, N3dContourLow, N3dContourInterval

These parameters set the high and low values in which constant lines are generated. the N3dContourInterval sets the interval between the high and low values at which the lines are calculated.

7.11 SURFACE

Surface plots are used to create a three-dimensional perspective of two-dimensional data with hidden lines removed. The surface, Z, is a function of two variables, X and Y.

For example, a three-dimensional surface could be created from a two-dimensional array of data where the first array dimension is the X position and the second dimension is the Y

position in a Cartesian grid. The value (Z) of each element in the array defines the height of the surface for each X and Y coordinate.

The following parameters define the characteristics of the surface such as rotation, surface representation, and viewing angle.

7.11.1 NsrArrayDims

This parameter specifies the X, Y dimensions of the data. This is an integer array of size 2.

7.11.2 NsrArray

This parameter is the data array which is dimensioned by NsrArrayDims.

7.11.3 NsrArrayRange

This parameter specifies the valid array range. It is an integer array of size 4. The first 2 array locations hold the lower left hand corner array indices, and the second 2 hold the array indices of the back right corner.

This allows the user to subsample data rather than use the entire data set.

The default value is the entire data set i.e. the array is 0 ,0 , xdim-1, ydim-1.

7.11.4 NsrArrayXstride, NsrArrayYstride

In a 2 dimensional data set it may be desirable to sample every other point or every third or fifth point for example. These parameters allow the user to specify the stride factor in each dimension.

Setting NsrufArrayXstride to 3 for example, forces this high level utility to sample every third data point in a the X direction.

The default is to sample every data point, therefore the parameters are initially set to 1.

By sampling down the data, the computations will run faster but your surface will be more coarse. This method of data downsizing is effective for preliminary viewing of data. Later, when exact viewing parameters are set, the user may want to view the entire data set.

7.11.5 NsrXcoord NsrYcoord

These are singly dimensioned arrays which contain the X and Y coordinates, respectively, of the points in a surface approximation. The units are in the same units as the data array.

These coordinate arrays allow the user to scale the data.

The size of the X coordinate array is the X dimension of the data array, and the size of the Y coordinate array is the Y dimension of the data array.

7.11.6 NsrEyepoint

This parameter specifies the xyz coordinate which determines the position of the viewers eye,.

This coordinate should be outside the data space.

7.11.7 NsrViewCenter

This parameter specifies the xyz coordinate which determines the point looked at by the eye. This coordinate should be inside the data space.

The distance between the eye and the point looked at should be 5 to 10 times the size of the data space block.

7.11.8 NsrStereo

This parameter is a flag to indicate if stereo pairs should be drawn.

A value of "on" indicates that the utility should draw stereo images, and a value of "off" indicates that it should not. "off" is the default.

When "off" is the flag value, the utility draws one image, when the flag is on, two images are drawn and are affected by the NsrStereoAngle parameter.

7.11.9 NsrStereoAngle

This parameter determines the relative angle between the eyes for stereo images. A value of 1.0 (the default) produces the standard separation. A negative angle reverses the images.

7.11.10NsrStereoType

This parameter determines how stereo images are placed on frames. The parameter may have 3 values: "Alternate", "Blank", or "Same".

"Alternate" specifies that the stereo images should be put on separate frames, but slightly offset.

"Blank" specifies that a blank frame should be created between the two stereo images. This is the default.

"Same" specifies that both stereo images should appear on the same frame.

7.11.11NsrPlotDirection

This parameter determines which plotting direction corresponds to the positive Z coordinates. This parameter may be either "Cine" or "Comic". "Cine" is the default. Changing these parameters will rotate the surface 90 degrees.

"Cine" specifies that +Z is in the vertical plotting direction.

"Comic" specifies that +Z is in the horizontal plotting directions.

7.11.12NsrConstLines

With this resource, users control the way the lines which represent the surface are drawn. The user can choose lines of constant U, V, W, or any combination thereof.

"U" Draw Lines of constant U.

"V" Draw Lines of constant V.

"W" Draw Lines of constant W.

"UV" Draw Lines of constant U and V.

"UW" Draw Lines of constant U and W.

"VW" Draw Lines of constant V and W.

"UVW" Draw Lines of constant U, V, and W.

"UV" is the default value.

7.11.13NsrDrSide

This parameter specifies which sides of a surface to draw.

"Upper" Draw upper side of surface.

"Both" Draw both sides.

"Lower" Draw lower side.

"Both" Default value.

7.11.14NsrSkirt

This flag specifies whether or not to draw a skirt around the object. A skirt is a wall which extends from the edge of the surface to a level defined by NsrSkirtBottom.

"on" Draw a skirt. "off" Do not draw a skirt.

"off" is the default value.

7.11.15NsrSkirtBottom

This is a real value which sets the level at which the bottom of a skirt (if NsrSkirt is on") terminates. It can be thought of as the height of the skirt.

0. is the default value

7.11.16NsrNLevels

This parameter determines the number of levels of constant Z. This must be an integer value greater than or equal to 0 and less than or equal to 40. The default is 6. The more levels you choose, the more dense the surface will be.

Note that you need to set NsrConstLines to include "W" so that levels of constant Z are drawn.

7.11.17NsrStereoTheta

This parameter sets the angle in radians between eyes for stereo pairs.

.02 is the default value.

7.11.18NsrContourLow NsrContourHigh NsrContourInterval

These parameters determine the highest and lowest levels of constant Z and the increment between them. 0. is the default value for all. These parameters allow the user to slice off part of the image in the Z direction and control the number of Z level contours.

If any of these parameters are 0., a nice value is generated automatically.

7.11.19NsrSpvalFlag

This parameter controls the use of NsrSpval.

"off" Do not use NsrSpval.

"on" Use NsrSpval. No lines are drawn to data points in Z that are equal to NsrSpval.

"off" is the default value.

7.11.20NsrSpval

This is the real data value used to mark special or missing data. If NsrSpvalFlag is "on", no lines are drawn to data points in Z that are equal to this parameter.

0. is the default value.

7.12 ISO-SURFACE

This section describes the resources which specify an iso-surface object. These objects are created by taking a 3D volume of data and specifying a threshold value. A polygon surface is created which intersects values in the data which equal the threshold value.

These surfaces or objects can then be operated upon. For example, a user can change the objects color, position, or lighting characteristics.

Besides iso-surfaces, this section describes the resources for 3D surfaces generated from 2D data (the z position is determined by the data value) and volumentric objects (data points are displayed as relative intensities where the intensity is proportional to the data value).

The 3D functionality is broken into two utilities: NCAR Graphics and PolyPaint.

The NCAR Graphics utility will be built upon the NCAR Graphics ISOSRF routine which generates iso-surfaces from a three-dimensional array.

The need to develop a programming interface to the PolyPaint application has not been confirmed. The interface presented here for PolyPaint is a possiblility of what the inteface may look like should it be determined that this is a high priority.

The PolyPaint utility will be built upon MMM's PolyPaint application which also generates iso-surfaces, but it includes support for color shading, volumentric rendering, and index and true color. PolyPaint also allows the user to control lighting, viewing, and shading.

The resources for 3D surfaces are divided into three sections: NCARGraphics iso-surface resources, common resources between NCAR Graphics Iso-surface and PolyPaint, and PolyPaint resources.

7.12.1 NCAR Graphics Iso-surface Resources

These resources are associated with the NCAR Graphics HLU and not the PolyPaint application.

7.12.1.1 NisConstLines

This parameter is used to determine which types of iso-surface lines to draw.

With this resource, users control the way the lines which represent the iso-surface are drawn. The user can choose lines of constant U, V, W, or any combination thereof.

"U" Draw Lines of constant U.

"V" Draw Lines of constant V.

"W" Draw Lines of constant W.

"UV" Draw Lines of constant U and V.

"UW" Draw Lines of constant U and W.

"VW" Draw Lines of constant V and W.

"UVW" Draw Lines of constant U, V, and W.

"UVW" is the default value.

7.12.1.2 NisVisibility

This parameter determines what data is inside and outside a surface.

"inside" - Data values greater than the threshold are assumed inside the surface.

"outside" - Data values less than the threshold are assumed outside the surface.

"inside" is the default value.

7.12.2 Common Resources between NCAR Graphics Iso-surface

and PolyPaint

The following resources are shared between the NCAR Graphics Iso-surface HLU and the PolyPaint application.

7.12.2.1 NisSurfaceData

This resource specifies the data array which defines the surface.

7.12.2.2 NisEyepoint

This is a floating point array with one dimension of size 3, and it holds the position of the eye in 3 space. The data is considered to be in a box with opposite corners at (1,1,1) and (xdim, ydim, zdim). The eye point should be outside of this box. NCAR Graphics documentation recommends (5*xdim, 3.5*ydim, 2.0*zdim).

Since the user's array size may be different for every application, a default value could be specified as a factor of the maximum array indices. i.e. 5.0, 3.5, 2.0 .

By changing the eyepoint, the user can adjust the angle or position from which the iso-surface is viewed. This allows the user to look at the object from any direction.

7.12.2.3 NisArrayDims

This parameter specifies the X, Y, and Z dimensions of the data. This is an integer array of size 3.

7.12.2.4 NisArrayRange

This parameter specifies the valid array range. It is an integer array of size 6. The first 3 array locations hold the front lower left hand corner array indices and the second 3 hold the back upper right array indices.

This allows the user to subsample data rather than use the entire data set.

The default value is the entire data set i.e. the array is 0 ,0 ,0 , xdim-1, ydim-1, zdim-1.

7.12.2.5 NisArrayXstride NisArrayYstride NisArrayZstride

In a 3 dimensional data set it may be desirable to sample every other point or every third or fifth point for example. These parameters allow the user to specify the stride factor in each dimension.

Setting NisArrayXstride to 3 for example, forces this high level utility to sample every third data point in a the X direction.

The default is to sample every data point, therefore the parameters are initially set to 1.

By sampling down the data, the computations will run faster but your iso-surface will be more coarse. This method of data downsizing is effective for preliminary viewing of data. Later, when exact viewing parameters are set, the user may want to view the entire data set.

7.12.2.6 NisAxes

This is an integer parameter to control whether or not the axes should be displayed.

"on" - Draw Axes "off" - Do Not Draw Axes

"on" - default value

7.12.2.7 NisThreshold

The parameter specifies the surfacer threshold. This value defines the iso-surface. A surface is drawn through points in a volume which equal this value.

The range must be a valid data value.

?????Min, or Max, Average, none - default value

7.12.3 PolyPaint Resources

The following resources or parameters and their definitions are taken from the Version 3.0 PolyPaint User Manual. These resources apply to the a proposed PolyPaint API that is consistent with the NCAR Graphics HLU concept.

Some of the resources apply only to index or true color settings. Index color uses a table of color values to determine the color of each point of an object. Normally a table has 256 entries. User's are allowed to divide a color table into several (up to 19) partitions. In this way, a separate partition can be used to color each object in a scene.

True color is different in that rgb values are used for each point of an object rather than an index into a color table.

How the final image is displayed depends on the color system of the display device. If the device is a true color device, it can display any color at any point on the screen. If the device is an index color device, there is a color table for that device, and it is used to display the image.

See appendix B of the PolyPaint User Manual Version 3.0 for a more complete description of the difference between index and true color.

7.12.3.1 NisAliasExpon

This parameter specifies the alias exponent. The floating point value sets the nonlinear color mixing exponent of anti-aliasing. It is a real value between 0. and 1.

.75 default value

The value will depend on the monitor used to display antialiased wire-frames. A value of 1 is linear.

7.12.3.2 NisAmbient

This parameter specifies the ambient light level intensity.

This can be either an intensity for index color or rgb values for true color.

0. or 0,0,0 are the default values.

7.12.3.3 NisXScale NisYScale NisZScale

These parameters specify the scale factors for each axis, and they allow the user to stretch and shrink an object in three directions.

1. is default value for each direction.

7.12.3.4 NisObjectBackIntensity

This parameter specifies an object's intensity as seen through a transparent object.

The parameter range is a real value between 0 and 1.

0. is the default value.

7.12.3.5 NisObjectFrontIntensity

This parameter specifies an object's transparency intensity. When this number is low and NisObjectBackIntensity is close to 1, the object looks very transparent; when this number is close to 1 and NisObjectBackIntensity is small, the object looks nearly opaque. This does not apply to thin-surface transparency.

The parameter range is a real value between 0 and 1.

1.0 is the default value.

7.12.3.6 NisPerpAxis

This parameters sets the axis which is perpendicular to the base plane. The base plane is a flat, rectangular object created with the position and orientation selected by a NisPerpAxis and NisBasePlaneLevel. The value may be the X, Y, or Z axes.

"Z" is the default value.

7.12.3.7 NisBasePlaneLevel

This parameter sets the level where the base plane is calculated. The level is <= nx, ny, or nz, depending on the perpendicular axis.

1 is the default value. ?

7.12.3.8 NisRedLimitMin NisRedLimitMax NisGreenLimitMin NisGreenLimitMax NisBlueLimitMin NisBlueLimitMax

These parameters set the range of brightness associated with the red, green, and blue raster-mapping arrays, respectively.

Min and Max real values between 0. and 1.

Min=0., Max=1. are the default values

7.12.3.9 NisViewCenter

This parameter specifies the xyz coordinates of an object's center. This is the position where the imaginary eye is looking. Its units are the same as the eye position.

.5,.5,.5 are the default coordinates.

7.12.3.10 NisPartitionNumber

This parameter specifies the number of color partitions a color table should be divided into. The more partitions a table is divided into, the fewer the number of colors in each partition.

This means that the dark to light difference from one index to another is greater.

This resource applies to index color only.

This is an integer value between 1 and 19. The default is 1.

7.12.3.11 NisColorSet

This parameter sets the color of the current partition when using index color or the surface color of the next object rendered if using true color.

If you are using index color, NisColorSet is a one dimensional array of size 6. The first 3 array locations hold the low values for red, green, and blue, and the last 3 locations hold the respective high values.

These values are real numbers between 0. and 1. 0.,0.,0., 1., 1., and 1. are the default values.

If you are using true color, NisColorSet is a one dimensional array of size 3. The first location holds a real number which specifies the amount of red in the object's surface color. The next two array locations hold the specifications for the amount of green and blue respectively.

These values are also real numbers between 0. and 1. 1., 1., and 1. are the default values.

7.12.3.12 NisCoordTrans

This parameter specifies the coordinate transformation. It can be either: cartesian, cylindrical, spherical, or auxiliary

Cartesian is the default value

7.12.3.13 NisCrossSectColorHigh NisCrossSectColorLow

NisCrossSectColorInterval

These resource is used to set parameters for cross section coloring for true color. Colors are assigned to a color ramp with the high end of the ramp assigned to NisCrossSectColorHigh and the low end assigned to NisCrossSectColorLow. If NisCrossSectColorInterval is 0, the coloring will be continuous, otherwise, constant colors will be assigned to ranges of width NisCrossSectColorInterval.

The defaults for the above three parameters are 0., 0., and 0..

7.12.3.14 NisCrossPartition

This parameter sets the color partition that the cross section will use for index color. It is an integer between 1 and the number of partitions available.

1 is the default value.

7.12.3.15 NisVolCutPlane

This parameter sets the level whereby values greater than the level are displayed if NisVolHigher is on, and values less than level otherwise.

0. is the default value

7.12.3.16 NisVolHigher

This is a toggle flag which is used in conjunction with the NisCutPlane parameter to determine what values of a surface to display.

The flag may be "on" or "off".

"On" is the default.

7.12.3.17 NisCuttingPlane

This integer parameter specifies the cutting plane in a data volume. The value specified determines the level at which to create a cutting plane.

The range is 1 to nx, ny, or nz depending on the cutting plane orientation.

7.12.3.18 NisAxisCutPlane

This parameter specifies the axis which is perpendicular to the cutting plane. The parameter may be X, Y, or Z corresponding to the X, Y, or Z axes respectively.

Z is the default value.

7.12.3.19 NisShadowDarkness

This parameter sets the darkness of the shadow which falls on objects. It is a real value between 0. and 1.

0. is the default value.

7.12.3.20 NisSpecularReflectionExponent

This real parameter determines the brightness of specular highlights. The larger the value the shinier the surface. Large exponents create little, bright highlights while small exponents create larger, more dispersed highlights.

5.0 is the default value.

7.12.3.21 NisSpecularIntensity

This parameter determines the intensity of specular highlights if using indexed color or the color if using true color.

It is a real value between 0. and 1. for indexed color, otherwise it is an RGB triple between 0. and 1. for true color.

0. (index) or 0.,0.,0. (true) are the default values.

7.12.3.22 NisThinSurfaceTransparency

This real parameter sets the value of the exponent for thin-surface transparency. The value of 0. is opaque.

1. is the default value.

7.12.3.23 NisTransTable

This parameter sets the transparency table for a current object, and is used to determine which objects are transparent to which other objects.

This is a two dimensional array where the first dimension contains an object number and the second dimension contains a toggle flag. 1 signifies that the current object is transparent to the object specified by object number. A 0 signifies that the current object is not transparent to the object specified by object number.

The default toggle flag is 0.

7.12.3.24 NisGeoTransformation

This parameter sets the coordinate transformation to use for geographical data.

Possible values are: Cartesian, wrap around globe, Equatorial Mercator Projection, or Sinusoidal projection

Cartesian is the default value.

7.12.3.25 NisHazeColor

This parameter specifies the haze color (true color) or brightness (indexed color) which the image fades to depending on distance from the eye.

If toggle autohaze is on, the background color is used as the haze color.

This is a single real value between 0. and 1. if indexed color is used. Otherwise, it is an RGB triple, between 0. and 1.

0 (indexed) or 0.,0.,0. (true) are the default values

7.12.3.26 NisInnerTransExponent NisOuterTransExponent

These two parameters define the inner and outer exponents used for the transparency level in nested shells. This allows the user to specify the relative intensities of the innermost and outermost shells. The exponents used for trnasparent shells vary linearly between the inner and outer exponents.

The parameters' ranges are real values between 0. and 1.

1. is the default.

7.12.3.27 NisLightNumber

This parameter specifies the number of lights in the scene. It is an integer value which is less than or equal to the maximum number of lights allowed.

1 is the default.

7.12.3.28 NisLightIntensity

This parameter sets the intensity for diffuse lighting of the current light source for index color. Otherwise, it specifies color of the current light source with an rgb triple.

Real value between 0. and 1. for index color RGB values between 0. and 1. for true color

1. (index), and 1.,1.,1. (true) are the default values.

7.12.3.29 NisLightIndex

This integer parameter specifies the current light source. It is a positive integer less than or equal to the maximum number of lights allowed.

1 is the default value.

7.12.3.30 NisLightPosition

This parameter specifies the coordinates of the current light. It is a one dimensional array of size three and holds the x, y, and z coordinates respectively.

10.,10.,10 is the default position.

7.12.3.31 NisMagnification

This parameter sets the amount of object magnification. It is a real value between 1.0 and 5.0

1.5 is the default value.

7.12.3.32 NisViewAngle

This real parameter sets the view angle for enlarging and shrinking an object. It is a real value between 0. and 90. The larger the value, the smaller the object will appear.

6.0 is the default value.

7.12.3.33 NisColorMidPoint

This parameter sets the midpoint value for the current color partition. Color partitions may vary nonlinearly by specifying the midpoint value of the partition. This is used for gamma correction.

Valid values are real numbers from 0. to 1.

The default is .5.

7.12.3.34 NisObjectIndex

This integer parameter specifies the current object identifier. The identifier ranges between 1 and 10.

1 is the default value.

7.12.3.35 NisPartitionIndex

This integer parameter sets the current color partition.

1 default value

7.12.3.36 NisTransOverlapColor

This parameter specifies an identifier of a color map to use for index color when pixels are overlapped by a transparent object. This is useful for mixing the colors of a front and rear object to create a realistic transparency effect.

1 is the default value.

7.12.3.37 NisRampNumber

This parameter specifies the number of color ramps for the true color option. It is an integer between 1 and 19.

1 is the default.

7.12.3.38 NisRampColor

This is an array of 7 real values. This array defines a linear color ramp for true color applications. The first 3 values are the low values for red, green, and blue. The next 3 values are the high values for red, green, and blue. The last value is the relative minimum of color data for the ramp.

Data from the color data array whose values range between the value specified by the relative minimum and the value of the relative minimum in the next color segment are colored using the linear coloring ramp.

0 is the default for each array value.

7.12.3.39 NisRampId

This parameter specifies the current data color-coding ramp.

1 is the default value

7.12.3.40 NisRotationAngle

This real parameter specifies the value of object rotation. The range of the angle is between 0. and 180.

15. is the default value.

7.12.3.41 NisShadowPosition

This real parameter specifies the Z position of a shadow cast onto a plane.

1.0 is the default value

7.12.3.42 NisShellLimitLow NisShellLimitHigh NisShellLimitInterval

These real parameters define the surface and intervals of shells. The shells are equal-valued surfaces for values ranging from NisShellLimitLow to NisShellLimitHigh at intervals of NisShellLimitInterval.

0., 0., 0. are the default values respectively.

7.12.3.43 NisVerticeLimit

This parameter specifies the maximum number of vertices a polygon can have. It is an integer between 3 and 7.

Limiting the vertices always increases the number of polygons and the demand for memory. For some purposes, such as hardware rendering, it may be necessary to limit the number of vertices per polygon.

7 is the default.

7.12.3.44 NisObjectActive

This is a toggle parameter which determines whether or not an object is active and will be rendered. The value may be "on" or "off".

"off" - do not render current object. "on" - do.

"on" is the default value.

7.12.3.45 NisDataAlternate

This parameter determines whether to use the color or main data for the cutting plane and cross section.

0 Use color data to color the cutting plane and cross section.

1 Use main data.

1 is the default value.

7.12.3.46 NisPixelAlternate

This specifies how to handle transparency. If this toggle is on, then objects will be superimposed by alternating pixels, otherwise the standard method will be used. Valid values are "on" or "off".

"off" is the default.

7.12.3.47 NisAntiAlias

This parameter toggles anti aliasing for shaded wire-frames. If it is on, the stair-stepped aliasing effect of a raster screen can be reduced. If the wire-frames are not shaded, they cannot be anti-aliased.

"on" is the default

7.12.3.48 NisArrayScale

This parameter toggles between scaling or not scaling the cutting- plane coloring by the values in the cutting-plane array.

"on" use array scaling "off" do not use array scaling

"on" is the default.

7.12.3.49 NisAutoHaze

This parameter toggles auto haze calculation. It will either be set automatically as the background color, or the user may manually enter the haze color.

"auto" - Automatically calculate haze color in true color
"manual" - Manually set haze coloring

"auto" is the default value

7.12.3.50 NisAutoCenter

This parameter toggles how to set the center of view.

"auto" - Automatically set the center of view "manual" - Manually set the center of view

"auto" is the default.

7.12.3.51 NisAutoScale

This parameter determines how the view angle is set.

"auto" - Automatically set the angle of view.

"manual" - Manually set the angle of view.

"auto" is the default.

7.12.3.52 NisSurfaceNormals

This parameter toggles between a smooth surface (surface normals are averaged) and a faceted surface (no averaging).

"smooth" - smooth surface "faceted" - faceted surface

"smooth" is the default.

7.12.3.53 NisBackfaceCull

This parameter toggles backface culling. This determines whether or not to render both sides of a surface. The parameter may be "on" or "off".

"on" - Render the backs of backward-facing polygons "off" - Do not

"off" is the default.

7.12.3.54 NisScaleBySlice

This parameter selects the option to scale cross-section coloring by the values in an array slice. The value may be "yes" or "no".

"on" - Scale cross-section coloring by the values in the array slice "off" - Do not

"off" is the default.

7.12.3.55 NisColorCoding

This parameter specifies whether or not to create and use color-coding data when creating polygons.

"on" - Use color-coding data "off" - Do not

"off" is the default.

7.12.3.56 NisSurfaceColorData

This parameter specifies the data to use for mapping color onto a surface. This is not a color table. This is the data used to determine which colors to use. The size of this array should be the same as NisSurfaceData.

7.12.3.57 NisShellTransparency

This parameter specifies whether or not to make transparent shells different colors, according to their contour values, using data color-coding ramps for true color, or make transparent shells different colors by assigning them consecutive partition numbers for index color.

"on" - Option is on

"off" - Option is off

"off" is the default

7.12.3.58 NisLightType

This parameter determines the current light's type.

"directional" - The current light source is directional only; light comes in from an infinite distance.

"point" - The current light source is a point source at a finite distance. The brightness of a point source decreases as the square of the distance.

"directionaL" is the default.

7.12.3.59 NisEyeRotation

This parameter toggles the ability to rotate the eye position.

"on" - Rotate eye position "off" - Do not

"on" is the default.

7.12.3.60 NisLightRotation

This parameter toggles the ability to rotate the light positions.

"on" - Rotate light position "off" Do not

"on" is the default.

7.12.3.61 NisShellLow

This parameter determines whether low or high values are assumed to be on the inside of a surface. This ultimately determines the order of rendering.

"on" Low values are assumed to be on the inside

"off" High values are assumed to be on the inside

"off" is the default

7.12.3.62 NisShading

This parameter specifies the shading algorithm.

"phong" - Use surface-normal (Phong) interpolation

"gouraud" - Use Gouraud shading

"gouraud" is the default.

7.12.3.63 NisSmallOpaque

This parameter allows the user to make the smallest shell in nested shells opaque or transparent.

"on" - Make the smallest shell opaque

"off" - Make the smallest shell transparent

"off" is the default.

7.12.3.64 NisShadowProjection

This parameter specifies how an object's shadow is projected.

"perpendicular" - Shadows are projected perpendicularly "bylight" - Shadows are projected according to the position of the light source

"bylight" is the default

7.12.3.65 NisCuttingPlaneColor

This parameter toggles between coloring and not coloring the interactive cutting plane.

"on" - Color it "off" - Do not color it

"off" is the default

7.12.3.66 NisSidewallPlanks

This parameter determines how the side walls are calculated. Side walls are the edges where a polygon intersects with the data field edge. See the NisSkirt resource below.

"planks" - Use long, plank-shaped polygons for side walls. this option is more efficient than "squares" but it may lead to rendering errors. "squares" - Use square polygons for side walls

"squares" is the default.

7.12.3.67 NisRasterScale

This parameter selects how to scale the red, green, and blue raster data arrays for true color.

"independent" - Scale the raster data arrays independently, scaling each array relative to its own max and min "

relative" - Scale the arrays relative the max and min for all three arrays

"relative" is the default

7.12.3.68 NisRasterMap

This parameter selects whether or not to superimpose raster data onto the surface of an object (true color option).

"on" - Superimpose raster data onto the surface "off" - Do not

"off" is the default

7.12.3.69 NisSelfTransparency

This parameter specifies whether or not the current object should be transparent to itself for index color. If this parameter is "on" any portion of the object being rendered that lies behind is treated as a back object to the portion of the object in front. If it is "off", any portion behind another portion is not seen.

"on" Make the current object transparent to itself "off" - Do not

"off" is the default.

7.12.3.70 NisWireShades

This parameter specifies whether or not to shade wire frames according to the current lighting model.

"on" - Shade wire frames "off" - Do not

"on" is the default

7.12.3.71 NisSurfaceFaceUp

This parameter determines whether or not surfaces face up surrounding higher values or down surrounding lower values.

"up" - Surfaces face up surrounding higher values "down" - Surfaces face down surrounding lower values

"up" is the default.

7.12.3.72 NisLightTerminator

This parameter selects between a 90 or 180-degree terminator for the current light source. A 180-degree terminator results in the most realistic lighting for one light source.

90 degrees produces harsh lighting, whereas 180 degrees provides a scattering effect similar to that of the atmosphere.

"180" degree "90" degree

"180" is the default

7.12.3.73 NisThinSurface

This parameter toggles between using or not using thin-surface transparency.

"on" - Use thin-surface transparency "off" - Do not

"off" is the default.

7.12.3.74 NisTrueColor

This specifies whether to use true color or index color calculations. On an 8-bit system, true color will be simulated by using a dithering algorithm.

"true" - Use true (24-bit) color "index" - Use index (8-bit) color

"index" is the default

7.12.3.75 NisVolhigher

This parameter specifies whether to display higher or lower values than the cutoff value (NisVolCutPlane) in volumetric rendering.

"high" - Display higher values "low" - Display lower values

"high" is the default

7.12.3.76 NisVolAlgorithm

This parameter selects the volumetric algorithm to use.

"quick" - Use the quick algorithm. This method is faster, but it can create striped effects when the data are seen from certain angles.

"slow" - Use the slow algorithm. This method is slower, but it uses a random-dithering algorithm which produces better results.

"slow" is the default

7.12.3.77 NisSkirt

This parameter specifies whether or not to build walls or endcaps to close off 3D contour surfaces when they intersect the edge of a data volume.

"on" - Build the walls "off" - Do not

"on" is the default

7.12.3.78 NisWireFrame

This parameter selects between wire frame or surface rendering.

"on" - Render objects as wire frames "off" - Render objects as surfaces

"off" is the default

7.12.3.79 NisBoundingBox

This parameter controls whether or not a bounding box is displayed around a data volume.

"on" - Turn on the bounding box "off" - Turn off the bounding box

"off" is the default

7.12.3.80 NisImageType

This parameter specifies how to render an image. It may be an iso-surface image, an iso-surface stereo image, a volumetric image, or a volumetric image in stereo.

"iso" - Render an iso-surface image

"stereo-iso" - Render an iso-surface stereo image

"volume" - Render a volumetric image

"stereo-volume" - Render a volumetric image in stereo

"iso" is the default.

7.12.3.81 NisZoomFactor

This parameter specifies how much to zoom in or out on an object.

Positive values cause an object to be rerendered larger, and negative values cause an object to be rerendered smaller.

7.12.3.82 Nis2DAxis

This parameter specifies the axis (X, Y, or Z) perpendicualr to the plane in which 2D contours or surfaces are generated.

"X" - Use the X axis

"Y" - Use the Y axis

"Z" - Use the Z axis

"Z" is the default

7.12.3.83 Nis2DContourLow Nis2DContourHigh Nis2DContourInterval

These parameters set the limits for 2D contours. Contours are generated from the data beginning with the value specified by the low argument up to the value specified by the high argument in increments specified by the interval argument.

0. is the default low value. 1. is the default high value. .1 is the default interval value.

7.12.3.84 Nis2DWidth

This parameter sets the width of 2D contours. The real value is a fraction of the data interval inthe dirction of the current 2D axis.

.3 is the default value.

7.12.3.85 Nis2DLevel

This specifies the level within the data volume at which 2D contours or surfaces are generated.

This is an integer between 1 and nx, ny, or nz, depending on the current 2D axis.

1 is the default.

7.12.3.86 Nis2DPlotLevel

This sets the level at which a 2D surface will be plotted.

It is an integer between 1 and nz.

1.0 is the default.

7.12.3.87 Nis2DSurfaceScale

This real parameter specifies the amount which data values are scaled in a 2D surface plot. In 2D plots, a polygon set is created from two-dimensional data in which z positions depend on data values.

1.0 is the default.

7.13 Annotation

7.13.1 LEGENDS

This section describes the resources which specify a legend object. Legends, like label bars (described below) are used as a key to accompany a plot. Legends are very similar to label bars, but they differ in that they define line representations (dash patterns) and symbols rather than fill patterns. Legends can be displayed vertically or horizontally, and labels may appear on any side.

7.13.1.1 NlgLegend

Turns on and of legend.

7.13.1.2 NlgOrientation

This integer resource specifies whether to create a vertical or horizontal row/column of legend symbols/lines. A value of 0 specifies horizontal and non-0 vertical. The default value is 0.

7.13.1.3 NlgSymbolOrientation

This integer resource specifies whether to create a vertical or horizontal symbol or dash-line pattern. A value of 0 specifies

horizontal and non-0 vertical. This resource is different from NiLegendOrientation in that it specifies the orientation of the individual symbols rather than the orientation relative to another symbol.

The default value is 0.

7.13.1.4 Nlgx Nlgy Nlgwidth Nlgheight

These real resources define the position coordinates of a rectangular area in which the legend, including its labels, are displayed. They are the X coordinates of the left and right edges of the area and the Y coordinates of the top and bottom edges respectively.

These real values are between 0. and 1. inclusive and represent a fraction of the plotter frame.

7.13.1.5 NlgElementType NlgNElements

Each element in the legend is either a dashed line or a symbol. This resource sets the type and NlgNelements keeps track of the total number of elements in the legend. These resources can be set automatically by the HLU. For example, an X-Yplot with mutiple curves fills in the legend with the dash patterns, symbols and curve labels it uses.

7.13.1.6 NlgIndex

This specifies the index of the current legend element when using an indexed scheme for defining legends. Legend attributes will only apply to the element selected by NlgIndex.

The default value is 1, the first element.

7.13.1.7 NlgDashPattern

Sets the dash pattern for elements that are line segments in the NlgElementType array.

NlgLineColor
NlgLineThickness
NlgDashLength

7.13.1.8 NlgSymbol

For elements that are symbols this sets the symbol attributes.

NlgSymbolScaleFactor
NlgSymbolColor
NlgSymbolFillPattern

The resource NlgIndex can be used to index individual symbols.

7.13.1.9 NlgLabelText

This is an array of strings which holds the labels which appear as annotation on the legend. The array size is NlgNsymbols, and there can be one label per symbol.

The labels have the standard "subresources" which are defined for generic text:

NlgLabelFont
NlgLabelFontSize
NlgLabelFontAspect
NlgLabelFontThickness
NlgLabelFontColor
NlgLabelTextJust
NlgLabelTextAngle

The resource NlgIndex can be used to index individual label text strings.

7.13.1.10 NlgLabelPosition

This integer resource specifies where the label sits with respect to its corresponding symbol.

0 symbol is unlabeled 1 labels are below a horizontal symbol or to the right of a vertical symbol

2 " "above " " left " "

3 labels are repeated on both sides of a symbol

7.13.1.11 NlgTitleText

This is the legend's title. This resource will have "subresources" such as those used to define generic text:

NlgTitleFont
NlgTitleFontSize
NlgTitleFontAspect
NlgTitleFontThickness
NlgTitleFontColor
NlgTitleTextJust
NlgTitleXOffset
NlgTitleYOffset
NlgTitleTextAngle

The default value is no title.

7.13.1.12 NlgDrawBorder

This resource is an integer flag which specifies whether or not a box should be drawn around the legend. A value of 0 specifies that a box should not be drawn, and a value not equal to 0 specifies that one should. 0 is the default value.

This resource will have "subresources" such as those used to define the generic line:

NlgBorderLineThickness
NlgBorderDashPattern
NlgBorderDashLength
NlgBorderLineColor
NlgBorderLineSmooth

7.13.2 LABELBARS

This section describes the resources which specify a label bar object. Label bars are used as a key to accompany a plot. They consist of a rectangular labeled bar which can be colored and filled solid or with patterned lines. Labelbars are different from Legends in that Legends define symbols and line representations, whereas Label bars define fill patterns.

A labelbar may be displayed vertically or horizontally and labels may appear on any side.

7.13.2.1 NlbLabelBar

Turns label bar on and off.

7.13.3 NlbOrientation

This integer resource specifies whether to create a vertical or horizontal bar. A value of 0 specifies horizontal and non-0 vertical. The default value is 0.

7.13.4 Nlbx Nlby Nlbwidth Nlbheight

These real resources define the position coordinates of a rectangular area in which the bar and its labels are displayed. They are the X coordinates of the top left edges of the area and the width and height of that area.

These real values are between 0. and 1. inclusive and represent a fraction of the plotter frame.

7.13.5 NlbNboxes

This is an integer resource which specifies the number or partitions or boxes into which the label bar is divided. All the boxes are of equal size and are filled with a pattern described below. Each box may have a unique color, pattern, and label.

The resource NlbBoxIndex can be used to index individual fill boxes.

7.13.6 NlbAreaWidth NlbAreaHeight

These resources specify what fraction of the label bar area is filled with a color or pattern. The unfilled area can be used for labels.

The range for these real resources is between 0. and 1. inclusive. The resources specify the fraction of the width and the height of the box, respectively.

7.13.7 NlbFillPattern

This integer resource specifies the pattern in the filled portion of a box. The fill pattern is referenced from the global fillpattern map.

7.13.8 NlbBoxColor

This specifies an integer array of indices into the color map described below. The size of the array is NlbNboxes. This array is used to determine the color of the box (solid color or line color depending on the fill pattern).

7.13.9 NlbLabelText

This is an array of strings which holds the labels which appear as annotation on the label bar. The array should be 1 of 3 sizes depending on how the user wants the labels to align with the boxes (below). If the alignment flag is 0, then the size is the same as the number of boxes in the bar. If the alignment flag is 1, the array size is one less than the number of boxes, and if the alignment flag is 2, then the size is one greater than the number of boxes.

The labels have the standard "subresources" which are defined for generic text:

NlbLabelFont
NlbLabelFontSize
NlbLabelFontAspect
NlbLabelFontThickness
NlbLabelFontColor
NlbLabelTextJust
NlbLabelXOffset
NlbLabelYOffset
NlbLabelTextAngle

The resource NlbBoxIndex can be used to index* individual text strings.

7.13.10NlbAlignment

This integer resource specifies how to align the labels to the boxes.

0 Align one label per box

1 Align labels with division between boxes

2 Align first label with the beginning of the bar, the last label with the end of the bar, and the rest of the labels to the divisions between the boxes.

*note that the label array (above) must be a different size in each case.

7.13.11NlbLabelPosition

This integer resource specifies where the label sits with respect to the box.

0 Bar is unlabeled

1 labels are below a horizontal bar or to the right of a vertical bar

2 " "above " " left " "

3 labels are repeated on both sides of a bar

7.13.12NlbBoxLineColor

The above resources determine the color of the label box and the labels respectively. Their values must be an integer color index greater than or equal to 0. The index maps into the color table described below.

NlbBoxLineThickness
NlbBoxLineDashPattern
NlbBoxLineDashLength

7.13.13NlbTitleText

This is the label bar's title. This resource will have "subresources" such as those used to define generic text:

NlbTitleFont
NlbrTitleFontSize
NlbTitleFontAspect
NlbTitleFontThickness
NlbTitleFontColor
NlbTitleTextJust
NlbTitleXOffset
NlbTitleYOffset
NlbTitleTextAngle

The default value is no title.

7.13.14NlbDrawPerim

This resource is an integer flag which specifies whether or not a box should be drawn around the labels and color bar. A value of 0 specifies that a box should not be drawn, and a value not equal to 0 specifies that one should. 0 is the default value.

This resource will have "subresources" such as those used to define generic line:

NlbPerimLineThickness
NlbPerimDashPattern
NlbPerimDashLength
NlbPerimLineColor

SECTION 8

GLOSSARY

1D:

Class of plot styles where is drawn by a line or symbol. Data is one dimensional array.

2D:

Class of plot styles where data points are in grid format. Data is a two dimensional cartesian grid.

3D:

Class of plot where data is defined with three spacial coordinates. Data is represented either as a three dimensional cartesian grid or random 3D points.

Annotation:

Any marking, line, text or fill area not belonging to a graph.

API:

Application Programmers Interface.

Control Panel:

A window that is displayed for the purpose of entering plot specifications graphically.

Color Map:

Array of Red, Green and Blue values. Used to specify colors used by a given instantiation of a Plot Style.

Coordinate Variable:

An NCL variable that has the same name and size as a dimension. This variable contains the coordinates for that dimension. For example, a coordinate variable for the dimension "longitude" contains a value in degrees longitude for each index in the dimension.

Coordinate Space:

Made up of all of the ranges of the coordinate variables of a varaible. The coordinate space for a lattitude by longitude grid over the whole world is [90..-90] x [180..-180]. This means that all data points of the variable are in this space. See Index Space.

DSMW:

Data Selection and Manipulation Window.

Data Sequence:

Input data as selected from the DSMW. The data is organized into spacial dimensions and timesteps. Each spacial dimension represents an axis on the output plot. The timesteps are other dimensions which are stepped through individually in a sequence. The Data Sequence describes how to step through the input data, in what order and over what ranges.

Device Independent Coordinates:

Coordinates of the Output Frame. They run from 0,0 in the upper left corner of the Output Frame to 1,1 at the lower right corner of the Output Frame

Fill Pattern:

Pattern which fills areas. Contours and other color regions can be filled with a pattern. Rather than just a color.

Font:

Text style.

Font Size:

Relative Size of font. Usually specified as percentage of Maximum Viewport Range.

Frame:

See Output Frame.

Graph:

Data plotted in a given plot style with associated tick marks, color bars and labels.

Graph Extent:

Bounding box in Device independent coordinates the contains the entire graph.

High Level Utility:

API for creating a Graph in a certain Plot Style.

Index Space:

This is related to Coordinate Space. Index Space is the integer index space defined by all of the dimensions of a variable. If a variable has 5 elements in one dimension and 6 in a second dimension he index space is 5x6. All data points lie with in these coordinates. Every variable has an index space.

Labels:

Any textual labeling belonging to a graph. (note. does not include annotation)

Label Bar: ‘

Bar containing Colors or Fill Patterns with Labels showing how color or Fill Patterns are mapped to data.

Layout and Display Screen:

For the GUI version of NI this is the area where the user can interactive specify the layout of the Output Frame. The user can size and position the plot viewports. When the Output Frame is rendered this area of the GUI becomes the Display Screen.

Line Thickness:

Relative thickness of a drawn line. Usually specified as percentage of Maximum Viewport Range.

Main Title:

Main Title for plot appearing above or below Viewport.

Maximum Viewport Range:

Largest of X and Y edges of Viewport.

NCL:

NCAR Command Language.

NCL Data File Record:

The main composite data type of the NCL language. It epresents one netCDF file of one or more variables. Each variable can have zero or more attributes associated with it.

NCL Variables:

Multidimensional variables in float, double, short, long or character data types.

Output Frame:

Window in which one or more graphs are plotted.

Plot:

The visual representation of data in a given Plot Style.

Plot Border:

Border drawn around the viewport. The Tick Marks extend from the plot border.

Plot Parameters:

Another term for Plot Resources

Plot Resources:

Piece of information used to configure the output of a high level utility.

Plot Specific Menu:

Each plot style has a unique set of menu for graphically specifying the visualization.

Plot Style:

Technique of visualization. XYPlot, Contour, Vector, Stream Line, Surface, Iso-Surface, CoPlot, 3D volume, etc...

Screen Coordinates:

Device Dependent coordinates for pixels on the screen.

Spacial Dimension:

Part of a data sequence. This is a dimension that is tied to an axis of the plot.

Static Dimension:

Part of a data sequence. It is a dimension that does not change.

Tick Mark:

Small line, drawn from the Plot Frame inward or outward, that marks a data value. Tick Mark are usually spaced at regular intervals along the Plot Frame.

Timestep:

See Timestep Dimension.

Timestep Dimension:

Part of a data sequence. This dimension is stepped through n some user specified fashion. Each step represents a new output frame.

Top Level Menu:

Main menu for GUI of NI.

User Coordinates:

World Coordinates.

Variable Attributes:

Descriptive information about the variable. Often the units of the variable, the range of the variable what values are missing values.

Viewport:

The portion of the output frame in which data is plotted. One or viewports can be specified for each output frame. An output frame with more than one viewport contains more than one graph.

Window:

Area of screen enclosing the Output Frame.

Window Coordinates:

These are the coordinates of the window with respect to Screen Coordinates.

World Coordinates:

Coordinates in data units. For 2D World Coordinates map onto the entire range of the Viewport. For 3D World Coordinates define a cube which is then projected on to the viewport.

X-axis Title:

Title for X-axis of 1d and 2d plots. Appears on top or bottom of viewport.

Y-axis Title:

Title for Y-axis of 1d and 2d plots. Appears on left or right of viewport.

Anti-aliasing:

A method for reducing the jagged appearance of lines.

Backface Culling:

This is an efficiency method which eliminates the processing of rear parts (polygons) of an object which are obstructed by the forward parts of an object. This effectively reduces the size of the final geometry.

Color Partition:

When using index color, color maps (usually with 256 entries) are used to color points on the screen. These color maps can be

divided into separated sections or partitions so that different objects can in effect have separate color maps. The more partitions a color map has, the less smooth the shading becomes.

Color Ramp:

This is the term used to describe a color table for true color systems. A color ramp is defined by a low red, green, and blue value; and a high red, green, and blue value. A linear color ramp is calculated between these values.

Endcaps:

See side walls.

Index Color:

Index coloring method uses a lookup table to determine the color of a point. Usually these tables contain 256 entries/ colors.

Light Terminators:

These are specified in degrees and can currently be either 90 or 180. These parameters determine at what angle the light has zero intensity. I.e. light terminates at either 90 or 180 degrees from the line of light, where the line is defined by a light's direction and its position. 90 degrees provides harsh lighting whereas 180 degrees provides a scattering effect similar to that of the atmosphere.

Nested Shells:

Sets of iso-surfaces which enclose one another. Nested shells are defined by minimum, maximum, and interval values. The minimum is the threshold for the inner-most shell, the maximum is the threshold for the outer-most shell, and the interval determines the spacing between shells.

Object and Object Identifier:

Objects are surfaces generated and displayed by PolyPaint. PolyPaint allows 10 objects to exist, each with a unique integer identifier.

Side Walls:

These are surfaces which can be turned on or off. These surfaces delineate the boundary of an iso-surface with the edge of a data volume. Also known as endcaps.

Specular Highlights:

Small areas of bright light which make an object look shiny.

True color:

The true coloring method stores the red, green, and blue values of each point on the screen in memory, where each point may take on a different color.

View Angle:

The angle which determines how narrow or broad the background appears in relation to the object being viewed. A small view angle specifies that the background is a small part of the displayed image, where as a large view angle includes a large portion of the background in a scene. Therefore, reducing the view angle effectively enlarges the image for a fixed size display window.

Volumetric Rendering:

This style of rendering displays points in a volume of data as relative intensities. The intensity of each point of data is proportional to the value of the data at that point.

SECTION 9 NCL SYNTAX

9.1 BASIC SYMBOLS

```
<basic_symbol> ::= <identifier> | <denotation> | <delimiter>
<identifier> ::= letter ((letter | digit)* ['_'] (letter | digit))*
        | letter ((letter | digit)* ['_'] (letter | digit))* '.' letter
                ((letter | digit)* ['_'] (letter | digit))*
        | letter ((letter | digit)* ['_'] (letter | digit))* '@' letter
                ((letter | digit)* ['_'] (letter | digit))*
        | letter ((letter | digit)* ['_'] (letter | digit))* '.' letter
                ((letter | digit)* ['_'] (letter | digit))* '@' letter
                ((letter | digit)* ['_'] (letter | digit))*
        | letter ((letter | digit)* ['_'] (letter | digit))* '.' letter
                ((letter | digit)* ['_'] (letter | digit))* '!' letter
                ((letter | digit)* ['_'] (letter | digit))*
        | letter ((letter | digit)* ['_'] (letter | digit))* '.' letter
                ((letter | digit)* ['_'] (letter | digit))* '&' letter
                ((letter | digit)* ['_'] (letter | digit))*

<letter> ::= 'A'-'Z' | 'a'-'z'
<digit> ::= '0' - '9'
<denotation> ::= <integer> | <real> | <string> | <array>
<integer> ::= <digit>+
<real> ::= <digit>+ '.' <digit>* | <digit>* '.' <digit>+
        | <real>'e'['+' | '-']<integer>
<string> ::= '"' <anychar> '"'
<array> ::= '[' <row_list> ']'
<row_list> ::= <row> | <row> ';' <row_list>
<row> ::= '[' (<expression> )(',' (<expression>))* ']'
<delimeter> ::= <special> | <keyword>
<special> ::= '^' | '*' | '#' | '/' | '.' | '+' | '-' | '<' | '>' | '='
```

```
        | ',' | ')' | '(' | '[' | ']' | ""
<keyword> ::= 'and' | 'or' | 'xor' | 'GT' | 'GE'
        | 'LE' |'LT' | 'EQ' | 'NE' | 'while' | 'do'
        | 'begin' | 'end' | 'then' | 'else' | <default_dims>
        | <typename> | 'return' | 'endo' | 'endwhile' | 'endif'
<default_dims> ::= 'ncl'(<digit>)+
<comment> ::= '/*' <arbitrary> '*/'
<typename> ::= 'long' | 'short' | 'float' | 'double' | 'character'
        | 'byte' | 'file' | 'numeric'
```

9.2 PROGRAM STRUCTURE

```
<program> ::= <block>
<block> ::= 'begin' <statement_list> 'end'
<statement_list> ::= <statement>
        | <statement> '\n' <statement_list>
<statement> ::= <assignment>
        | <if>
        | <block>
        | <procedure>
        | <loop>
        | <visblk>
<loop> ::= <do>
        | <while>
<do> ::= 'do' <var> '=' <expression> ',' <expression> [','
         <expression>] <statement_list> 'endo'
<while> ::= 'while' <expression> <statement_list> 'endwhile'
<if> ::= 'if' <expression> 'then' <statement_list> 'endif'
        | 'if' <expression> 'then' <statement_list> 'else' <statement>
         'endif'
<procedure> ::= <name>
        | <name> '(' <parameter_list> ')'
<proc_def> ::= 'procedure' <name> '('<para_def_list>')' <block>
<para_def_list> ::= <name> [<dimsizelist>] ':' <typename>
```

| <name> [<dimsizelist>] ':' <typename> ','
<para_def_list>
<dimsizelist> ::= '[' digit+ ']' | '[' '*' ']'
| '[' digit+ ']' <dimsizelist>
| '[' '*' ']' <dimsizelist>
<parameter_list> ::= <expression>
| <expression> ',' <parameter_list>
<visblk> ::= <utilityname> <plotname> '{' <anychar> '}'
<plotname> ::= <identifier>
<utilityname> ::= <identifier>

9.3 EXPRESSIONS

<expression> ::= <assignment>
| <disjunction>
<assignment> ::= <name> '=' <expression>
<disjunction> ::= <conjunction>
| <disjunction> 'OR' <conjunction>
| <disjunction> 'XOR' <conjunction>
<conjunction> ::= <comparison>
| <conjunction> 'AND' <comparison>
<comparsion> ::= <relation>
| <relation> <eqop> <relation>
<eqop> ::= 'EQ'
| 'NE'
<relation> ::= <sum>
| <sum> <relop> <sum>
<relop> ::= 'LE'
| 'LT'
| 'GT'
| 'GE'
<sum> ::= <term>
| <sum> <addop> <term>
<addop> ::= '+'
| '-'

```
        | '<'
        | '>'
<term> ::= <expfactor> | <term> <mulop> <expfactor>
<mulop> ::= '*'
        | '#'
        | '/'
        | '%'
<expfactor> ::= <factor>
        | <expfactor> <expop> <factor>
<expop> ::= '^'
<factor> ::= <primary>
        | <unop> <factor>
<unop> ::= '+'
        | '-'
        | 'NOT'
<primary> ::= <denotation>
        | <name>
        | '(' <expression> ')'
        | <block>
        | <function>
<name> ::= <identifier>
        | <identifier> <subscript_list>
<subscript_list> ::= <subscript>
        | <subscript> <subscript_list>
<subscript> ::= '[' <expression> ']'
        | '[' <expression> ':' <expression> ']'
        | '[' '*' ']'
        | '[' '*' ':' '*' ']'
        | '[' '*' ':' <expression> ']'
        | '[' <expression> ':' '*' ']'
        | '{' <expression> '}'
        | '{' <expression> ':' <expression> '}'
        | '{' <expression> ':' '*' '}'
        | '{' '*' ':' <expression> '}'
        | '{' '*' ':' '*' '}'
        | '{' <identifier> '|' <expression> '}'
        | '{' <identifier> '|' <expression> ':' <expression> '}'
```

| '{' <identifier> '|' <expression> ':' '*' '}'
| '{' <identifier> '|' '*' ':' <expression> '}'
<function> ::= <name> '(' <parameter_list> ')'
<fun_def> ::= 'function' <name> '('<para_def_list>')' <block>

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Contractor Report

4. TITLE AND SUBTITLE

Interactive Interface for NCAR Graphics

5. FUNDING NUMBERS

930

6. AUTHOR(S)

Bill Buzbee, Bob Lackman, Ethan Alpert

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Center for Atmospheric Research
P.O. Box 3000
Boulder, CO 80301

8. PERFORMING ORGANIZATION REPORT NUMBER

NAS5-32337
5555-15

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration - OSSA
Washington, D.C. 20546-0001

Universities Space Research Association
10227 Wincopin Circle, Suite 212
Columbia, MD 21044

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CR-189383

11. SUPPLEMENTARY NOTES

Technical Monitor: J. Hollis, Code 930

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified–Unlimited
Subject Category 82
Report is available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The NCAR Graphics package has been a valuable research tool for over 20 years. As a low level Fortran library, however, it was difficult to use for non-programming researchers. With this grant and NSF support, an interactive interface has been created which greatly facilitates use of the package by researchers of diverse computer skill levels.

14. SUBJECT TERMS

Interactive graphics, NCAR command language, High-level Utilities

15. NUMBER OF PAGES

226

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited